DB2REST: Uma Solução para Integração entre Bases de Dados Legadas e Web Services RESTful

Bruno A. Oliveira[1], **Sandro S. Andrade**[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia da Bahia (IFBA)
Salvador – BA – Brasil

{brunoaraujo,sandroandrade}@ifba.edu.br

Abstract. *The integration between computational systems implies challenges commonly related to incompatible interfaces and technologies. A particular case of this difficulty is the integration between RESTful web services and legacy databases, whose data structures are divergent. In this context, this paper aims to present DB2REST, a software solution that assists in the integration between RESTful web servers and legacy databases. To evaluate the solution, experiments were performed with and without the DB2REST, aiming to analyze the productivity, complexity and density of bugs in each scenario.*

Resumo. *A integração entre sistemas computacionais implica em desafios comumente relacionados a interfaces e tecnologias incompatíveis. Um caso particular dessa dificuldade é a integração entre web services RESTful e bases de dados legadas, cujas estruturas de dados são divergentes. Neste contexto, este trabalho apresenta o DB2REST, uma solução de software que auxilia na integração entre servidores web RESTful e bases de dados legadas. Para avaliar a solução foram realizados experimentos com e sem o DB2REST, visando analisar a produtividade, complexidade e densidade de bugs em cada cenário.*

1. Introdução

Ao longo das últimas décadas, com o advento dos Sistemas de Informação, a tecnologia passou a ser um agente catalisador de profundas mudanças nas organizações, influenciando desde a forma como são administradas e até mesmo o local de realização do trabalho [Galvão et al. 2009]. Tais soluções de software muitas vezes são desenvolvidos por diferentes fabricantes e tecnologias, seguindo uma lógica de departamentalização que está ligada a hierarquia empresarial. Desta forma, faz-se necessário desenvolver mecanismos de integração entre tais sistemas.

Existem diversas situações em que a integração entre sistemas é requerida. As empresas costumam investir muito dinheiro no desenvolvimento de softwares e, para obter o retorno sobre este investimento, é necessário que o software seja utilizado por vários anos. Neste contexto, desenvolver novas soluções a cada nova demanda muitas vezes demonstra-se inviável [Pinto and Braga 2005] [Lapolli 2003]. Alguns casos em que comumente existe a necessidade de integração são: quando têm-se bases de dados legadas; quando existe a necessidade de interoperar com clientes de múltiplas plataformas; quando é preciso expor serviços na web ou usar plataformas de *cloud computing*; quando têm-se B2B (*Business-to-business*).

Um caso particular dos principais desafios de integração entre sistemas está na adaptação entre *web services RESTful* e bases de dados legadas. Considere a existência de dois cenários: no primeiro, uma aplicação para dispositivos móveis que permite a visualização da programação de um evento. A aplicação móvel consome dados de um *web services RESTful* para exibir informações sobre os eventos, atividades realizadas e suas locações, horários, dados sobre os facilitadores, etc. Este *web services RESTful* foi construído considerando uma configuração de dados específica, isto é, ela considera tabelas, colunas e relacionamentos sobre os quais os serviços são implementados. No segundo cenário, considere a existência de uma instituição que realiza eventos periodicamente, cujos dados são organizados da forma que a instituição julgou necessária. Agora, supondo que a instituição do cenário 2, que já possui uma estrutura de dados previamente definida, deseje adotar a aplicação móvel definida no cenário 1 para divulgar a ocorrência dos eventos por ela realizados, porém não deseje alterar sua forma de organizar os dados. Como adaptar essas interfaces de forma que ambas as aplicações possam manter suas características?

Algumas soluções de ORM (*Object-Relational Mapping*), como o Django *Framework* e o SQLAlchemy, permitem construir aplicações em torno de bases de dados existentes. Tais soluções permitem gerar o código no padrão esperado pela ferramenta a partir de uma base legada. Entretanto, elas pressupõe a criação dessas APIs a partir da estrutura do banco de dados, não havendo suporte para adaptação nos casos em que a *API RESTful* já existe. Neste contexto, o presente trabalho tem como objetivo a implementação e avaliação de uma solução flexível para facilitar a integração entre *web services RESTful* e bases da dados legadas, o DB2REST. Este componente de software realiza a adaptação entre a camada de serviço e a camada de persistência, permitindo a integração entre esses componentes sem a necessidade de modificação da *API RESTful* existente ou do *schema* da base de dados legada, reduzindo assim os problemas de implementação e implantação destes componentes.

Para avaliar a solução proposta nesse trabalho, foram realizados experimentos em dois cenários distintos: no primeiro, um novo *web service RESTful* foi desenvolvido; no segundo, utilizou-se o DB2REST para realizar a integração entre um *web service RESTful* existente e uma base de dados legada. A partir da análise desses experimentos, foram extraídas métricas de software para avaliar a efetividade da solução.

Além desta introdução, este trabalho está organizado como se segue. A Seção 2 apresenta os trabalhos relacionados à solução proposta neste trabalho. A Seção 3 apresenta a solução proposta e as tecnologias utilizadas no seu desenvolvimento.

2. Trabalhos Correlatos

Existem diversas soluções que visam prover flexibilidade da camada de persistência em sistemas computacionais. Dentre as soluções mais relevantes está o uso da técnica de ORM [Coelho and Sartorelli 2004], atualmente implementado em vários *frameworks*, que consiste em realizar o mapeamento de objetos em memória para meios de persistência relacionais de forma transparente ao usuário [Płuciennik-Psota 2012]. Os *frameworks* considerados mais relevantes serão discutidos a seguir.

O SQLAlchemy (https://www.sqlalchemy.org/) é um *toolkit* e *framework* para ORM, desenvolvido para a linguagem Python. Tem como principais características sua extensibilidade e o fato de ser um ORM independente, o que torna

mais fácil adequá-lo às necessidades de um projeto [Gantan 2014]. O *Sandman2* (http://sandman2.readthedocs.io), o Sqlacodegen (https://github.com/ksindi/flask-sqlacodegen) e o Flask-RESTful (https://flask-restful.readthedocs.io) são extensões para SQLAlchemy que possibilitam a geração automática de serviços RESTFul e/ou modelos esperados pelo SQLAlchemy. Essas extensões apresentam como ponto forte prescindir de nenhuma configuração adicional para funcionar. Entretanto, apresentam algumas desvantagens: dispõe de pouquíssimas opções de personalização da API REST (*Sandman2*), a ocorrência de *overhead* ao processar o código gerado dinamicamente (*Sandman2* e *Sqlacodegen*) e a falta de suporte para bases de dados legadas (*Flask-RESTful*).

O Django (https://docs.djangoproject.com) é um *framework* web *full-stack*, também escrito em Python, que possui uma implementação própria da técnica de ORM. Além disso, disponibiliza uma ferramenta chamada *inspectdb*, que facilita a integração com bases de dados legadas a partir da geração automática de modelos introspectando um banco de dados existente. O Django ORM apresenta como desvantagens o fato de funcionar apenas integrado ao Django.

Além do SQLAlchemy e Django ORM, têm-se um outro *framework* desenvolvido para a linguagem Python, chamado SQLObject (http://sqlobject.org/SQLObject.html). Apesar de possuir baixa curva de aprendizado, possui várias limitações em relação aos *frameworks* supracitados, limitando o seu uso a projetos de pequeno porte. O Ruby on Rails (RoR) (http://guides.rubyonrails.org/) é um *framework* de desenvolvimento web que utiliza a linguagem orientada a objetos Ruby. Uma das bibliotecas do RoR, o *ActiveRecord* é uma camada de ORM que é responsável pela interoperabilidade entre os banco de dados.

O Hibernate (http://hibernate.org/orm/) é *framework* de ORM escrito em linguagem Java, que abstrai o seu código SQL, toda a camada JDBC e SQL gerado em tempo de compilação. Tem como objetivo de diminuir a complexidade entre os programas desenvolvidos em Java que precisam trabalhar com banco de dados relacional.

Embora tenham sido apresentado algumas ferramentas que proveem flexibilidade de banco de dados, existem alguns desafios inerentes a integração com bases de dados legadas, como a integração dessas bases à serviços *RESTful* já existentes, cujas interfaces são diferentes da estrutura da base de dados. Nesse contexto, este trabalho propõe uma solução de software para facilitar essa integração, de forma que seja possível realizar a introspecção de uma base existente e o mapeamento para uma interface *RESTful* existente.

3. A Solução desenvolvida

Nesta seção será apresentado o DB2REST. A subseção 3.1 apresenta as informações e o diagrama da estrutural da solução. Em seguida, a subseção 3.2 elucida, de forma detalhada, como preencher cada campo do arquivo de configuração. Por fim, a subseção 3.3 apresenta os aspectos de implementação da ferramenta.

3.1. Arquitetura do DB2REST

A ferramenta apresentada neste artigo visa adaptar a interface de um *web services RESTful* — cujos serviços implementam regras de negócio que pressupõe a existência de uma estrutura de dados específica, como entidades e relacionamentos, sobre os quais os serviços

realizam algum processamento — e uma base de dados legada, utilizando para isto um arquivo de especificação em formato JSON, contendo a descrição dos modelos de dados da base legada e do *web services RESTful* e de seus respectivos atributos e relacionamentos.

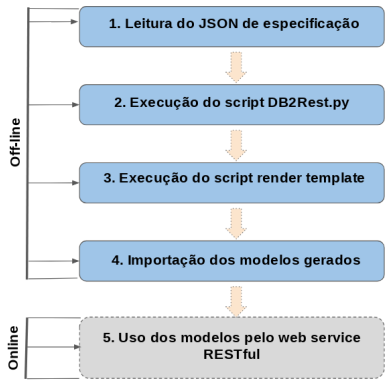

Figura 1. Fluxo de execução do DB2REST

A Figura 1 apresenta o fluxo de execução do DB2REST. A ferramenta possui duas etapas de execução distintas: uma parte é executada offline (1 - 4) e a outra, online (5). A seguir, têm-se uma breve descrição de cada uma dessas etapas:

1. **Leitura do Json de Especificação** — que será descrito detalhadamente a seguir— contendo as informações requeridas pela ferramenta;
2. **Execução do *script* db2rest.py** que é o mecanismo principal da ferramenta. Nele, são verificadas e validadas todas as informações do JSON de especificação na base de dados e a criação da estrutura do código que será gerado pela ferramenta;
3. **Execução do *script render_template***, que é um mecanismo auxiliar para geração de código a partir de um *template*;
4. **Importação dos modelos gerados**, consiste na configuração para uso dos modelos gerados pelo *web service RESTful*.
5. **Uso dos modelos gerados pelo *web service RESTful*** é a única etapa do modo de execução online. Os modelos gerados nas etapas anteriores são utilizados pelo servidor. Para cada modelo no arquivo de especificação é gerada uma classe, em linguagem Python, que é a representação daquela entidade no banco de dados e, ao mesmo tempo, realiza a adaptação entre a estrutura do banco de dados e a estrutura do *web service RESTful*.

3.2. Definições do Arquivo de Configuração

O arquivo de configuração em formato JSON deverá conter um registro para cada modelo a ser adaptado. Além disso, são necessárias informações sobre a entidade-alvo na base de dados legada, bem como outras informações sobre seus atributos, relacionamentos e outras propriedades. A seguir, serão descritas as propriedades do arquivo de configuração e uma breve descrição da sua utilidade. Os atributos com prefixo *rst* representam os parâmetros do *web service RESTful*, e aqueles iniciados com o prefixo *db* referem-se às informações da base de dados. A Listagem 1 apresenta um exemplo do arquivo de configuração esperado pelo DB2REST.

```
[{
"__rst_model_name__": "Postagem", "__db_table_name__": "post",
"attributes": [
      {"rst_attribute_name": "id_postagem","db_column_table":"id","db_primary_key": "True"},
      {"rst_attribute_name": "titulo","db_column_table":"title"},
      {"rst_attribute_name": "data_postagem","db_column_table":"date"},
      {"rst_attribute_name": "hora_postagem","db_column_table":"time" }],
"derived_attributes":[{
    "rst_property_name": "detalhes_categoria","db_columns": "category.name",
    "db_clause_where": "id|1","db_rows_many": "False"}],
"relationships": [{
    "type":"M2O", "rst_referencing_name": "categoria","rst_referenced_model":"Categoria",
    "db_referenced_table":"category", "db_referenced_table_pk":"category.id",
    "db_referencing_table_fk": "category","rst_referenced_backref": "postagens"}]
}]
```

Listing 1: Exemplo do arquivo de configuração em formato JSON

- ***__rst_model_name__***: deverá conter o nome do modelo de dados utilizado pelo *web service RESTful*, que neste caso representa um recurso.
- ***__db_table_name__***: deverá conter o nome da entidade correspondente na base de dados legada.
- ***attributes***: deverá conter uma lista de atributos a serem mapeados. Para cada atributo, têm-se um objeto JSON que contém informações sobre o respectivo atributo. Cada chave dentro desse objeto representa uma propriedade que será aplicada aos modelos gerados. A definição desta estrutura para cada atributo permite a inclusão futura de novas informações a serem mapeadas.
- ***derived_attributes***: deverá conter uma lista de objetos JSON para cada atributo derivado a ser gerado. Esta chave tem como finalidade obter informações sobre atributos presentes em outras entidades, e que serão mapeados como atributo do modelo, mesmo que não exista relacionamento entre eles na base de dados. Sua utilização é indicada quando alguma informação requerida pelo *web service RESTful* está presente em outra tabela da base de dados.
- ***relationships***: deverá conter uma lista de objetos JSON, com um registro para cada relacionamento a ser mapeado. Esta chave deverá ser preenchida considerando a perspectiva do modelo atual em relação ao modelo-alvo.

3.3. Aspectos de Implementação

O DB2REST foi desenvolvido utilizando como base o SQLAlchemy, um *framework* que implementa a técnica de ORM e tem como principais vantagens o fato de ser bastante flexível e independente. No contexto do DB2REST, o SQLAlchemy é utilizado para realizar essa introspecção na base de dados legada e, desta forma, realizar a integração com os modelos de dados do *web service RESTful*.

Após a criação do arquivo de configuração, o DB2REST realiza a leitura do arquivo e valida as informações na base de dados, checando os nomes das tabela, atributos e relacionamentos informados. Caso não sejam encontradas inconsistências no arquivo de especificação, a ferramenta gera um conjunto de dados sobre os modelos que serão delegados a um *template*. O *template* utilizado pelo DB2REST é desenvolvido usando o *Jinja2*, uma linguagem de *templates* para a linguagem Python.

```python
from DB2Rest.db import Base
from sqlalchemy import Column,Integer,String,ForeignKey
from sqlalchemy.orm import relationship
from sqlalchemy.ext.hybrid import hybrid_property
from DB2Rest import queries

class Postagem(Base):
    __tablename__ = "post"
    id_postagem = Column('id',Integer,primary_key=True)
    titulo = Column('title')
    data_postagem = Column('date')
    hora_postagem = Column('time')

    ##Relationships##
    categoria_id = Column('category',Integer, ForeignKey('category.id'))
    categoria = relationship('Categoria', back_populates='postagens', lazy='joined')

     @hybrid_property
    def detalhes_categoria(self):
        return queries.get_table_derived_attributes(table_name='category', column_name='name',
            clause_where_attribute='id',clause_where_value=1, many=False)
```

Listing 2: Classe gerada automaticamente pelo DB2REST

As classes geradas herdam de uma classe *Base*, provida pelo DB2REST, e configurada de acordo com a especificação de API do SQLAlchemy para mapear uma base de dados existente. Cada classe é uma representação na linguagem Python, para cada entidade na base de dados legada. Estas classes são interpretadas pelo SQLAlchemy como parte da aplicação. A Listagem 2 apresenta um exemplo de classe gerada automaticamente pela ferramenta.

A Listagem 3 demonstra como os modelos gerados pelo DB2REST são utilizados em um *web service RESTful*. O serviço, disponível na URL *postagens* utiliza o modelo Postagem, apresentado na Listagem 2, a partir da importação do arquivo *models* do módulo DB2REST (linha 5). O serviço retorna uma lista dos registros de postagens, utilizando o formato padrão da API do SQLAlchemy para fazer a consulta na base de dados. Ao acessar o modelo Postagem, os métodos assessores farão o mapeamento das entidades e atributos do modelo.

```python
from flask import Blueprint, jsonify, request
import datetime
from DB2Rest import models

postagem = Blueprint("postagem", __name__)

@postagem.route('/postagens')
def listar_postagens():
    postagens = models.Postagem.query.all()
    return jsonify(result=[dict(id=postagem.id_postagem,titulo=postagem.titulo,
            data=postagem.data_postagem,hora=postagem.hora_postagem,
            categoria=postagem.categoria,detalhes=postagem.detalhes_categoria)
            for postagem in postagens])
```

Listing 3: Serviço RESTful que utiliza o mecanismo DB2REST

4. Avaliação e Resultados

Nesta seção serão apresentadas informações relativas aos experimentos realizados com usuários, com o objetivo de testar e avaliar a solução proposta nesse trabalho.

4.1. Estudo de Caso: Emile Server

O Emile é sistema *open-source* para comunicação acadêmica, desenvolvido no âmbito do GSORT (Grupo de Sistemas Distribuídos, Otimização, Redes e Tempo Real), grupo de pesquisa do IFBA (Instituto Federal da Bahia). O sistema foi projetado para permitir a aproximação entre a academia e os seus diversos atores através de um aplicativo móvel.

O sistema é composto pelo aplicativo móvel propriamente dito e de um *web service RESTful*, o *Emile Server*, que provê dados para consumo no aplicativo. No início do projeto, a equipe de desenvolvimento não obteve acesso à base de dados legada devido a questões burocráticas e procedimentos de segurança da informação adotados pela instituição. Por isso, os desenvolvedores criaram uma nova modelagem de dados, em conformidade com as entidades identificadas pela equipe naquele período.

4.2. Execução dos Experimentos

Para avaliar a solução proposta neste trabalho, foi conduzido um estudo com dois participantes (P1 e P2), ambos alunos do curso de Análise e Desenvolvimento de Sistemas do Instituto Federal da Bahia (IFBA) - Campus Salvador. A execução dos experimentos foi precedida de um mini-treinamento, com 2 horas de duração, no qual os participantes aprenderam sobre as tecnologias que seriam utilizadas no experimento, visando nivelar os conhecimentos sobre os conteúdos abordados. A execução do estudo ocorreu nos dias 7, 8 e 14 de março de 2018, no laboratório do GSORT (Grupo de Sistemas Distribuídos, Otimização, Redes e Tempo Real). Os experimentos tiveram duração mínima de 1 hora e máxima de 3 horas.

Os experimentos foram realizados utilizando o código-fonte do *Emile Server*. Para isso, um conjunto de modelos e serviços foram escolhidos como base para utilização em dois cenários diferentes. Ambos os participantes foram submetidos aos experimentos I e II, sendo que a ordem de execução destes se deu de forma inversa. Isto é, um dos participantes começou pelo experimento I e o outro pelo experimento II. No primeiro experimento, foi disponibilizada uma versão do código-fonte sem a implementação dos modelos e serviços. No segundo experimento, foi disponibilizada uma versão contendo as implementações dos modelos e serviços, porém com uma base de dados diferente. Os participantes utilizaram o DB2REST para a integração.

4.3. Resultados e Discussão

O presente estudo visa analisar e avaliar os resultados obtidos na execução dos experimentos I e II. Os códigos desenvolvidos pelos participantes dos experimentos foram enviados para o repositório de testes para análise e extração de métricas de software. A Tabela 1 apresenta as hipóteses que serão analisadas com base nos resultados obtidos.

Tabela 1. Hipóteses Avaliadas no Estudo

Hipóteses	Métricas	Resultados Esperados
H1	Produtividade: tempo	$T_{DB2REST} < T_{Flask}$
H2	Complexidade: LOC	$LOC_{DB2REST} < LOC_{Flask}$
H3	Densidade de bugs: bugs/LOC	$DB_{DB2REST} < DB_{Flask}$

Para analisar H1, o tempo de execução gasto para desenvolver cada etapa foi computado no estudo. A Figura 2 apresenta um gráfico com as informações de tempo decorrido em cada experimento. Os pontos no gráfico representam os *check points* de cada experimento, divididos em duas etapas. Observa-se que os participantes demandaram menos tempo na execução das etapas relativas ao experimento II em relação ao experimento I. Tal característica deve-se ao fato de que, no caso do experimento II, não foi necessário implementar a lógica de negócio necessária para o funcionamento dos serviços, bastando apenas mudar o módulo de importação.

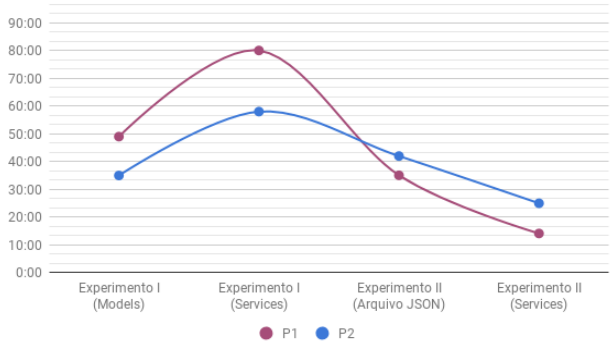

Figura 2. H1: Produtividade em função do tempo de desenvolvimento

Considerando os dados demonstrados no gráfico da Figura 2, e com base nas análises realizadas, pode-se considerar que H1 é verdadeira. A condição $T_{DB2REST} < T_{Flask}$ é válida, uma vez que integrar um *web service RESTful* a uma base de dados legada consome menos tempo, e portanto, é mais produtivo do que desenvolver um novo serviço.

A avaliação de H2 compreende o uso da métrica *LOC* (*Lines of Code*) para analisar a complexidade envolvida em: i) reimplementar os serviços RESTful para uma nova base de dados ii) integrar serviços existentes a uma base de dados legada. Para melhor compreensão dos dados analisados, os modelos e serviços desenvolvidos nos experimentos foram separados em dois gráficos, sendo um deles apenas para os modelos e o outro apenas para os serviços escolhidos para a execução dos experimentos.

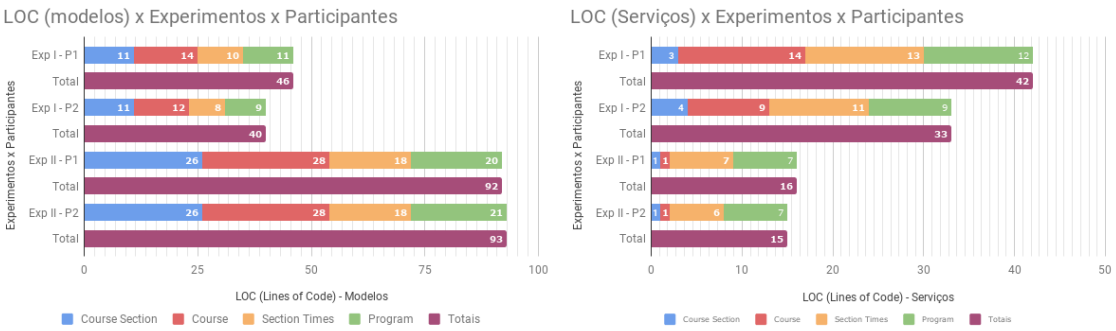

Figura 3. H2: Análise de Complexidade Modelos/Serviços

A Figura 3 apresenta os gráficos relativos ao quantitativo de linhas escritas pelos participantes para cada modelo e serviço, acompanhados de uma coluna com a quanti-

dade total de linhas para cada experimento/participante. No primeiro gráfico da Figura 3, referente aos modelos, observa-se que a quantidade de linhas necessárias no experimento II, utilizando o DB2REST, é o dobro da quantidade de linhas requeridas no experimento I. Entretanto, um fator preponderante nesta análise refere-se à estrutura do arquivo de configuração que pela própria definição da notação JSON, acrescenta linhas ao arquivo e, por sua vez, influenciam diretamente na quantidade total do *LOC*. O segundo gráfico disponível na Figura 3 demonstra o *LOC* apenas para os serviços implementados na execução dos experimentos. Nota-se que a quantidade de linhas necessárias no experimento II foi consideravelmente menor do que a quantidade requerida no experimento I, representando a metade da quantidade de linhas utilizadas no primeiro cenário.

A partir da análise dos gráficos disponíveis nas Figura 3, conclui-se que H2 é falsa. O LOC dos experimentos indicam que $LOC_{DB2REST} < LOC_{Flask}$ não é verdadeira. Dessa forma, utilizar o DB2REST para integrar um *web service RESTful* a uma base de dados legada requer mais linhas – e portanto, gera sistemas mais complexos – do que desenvolver um novo *web service RESTful*. Contudo, é importante salientar que o presente estudo identificou que a métrica utilizada não é a mais adequada para avaliar os cenários disponíveis, devido a discrepância entre os tipos de código analisados.

Para fins de avaliação da H3, foi realizada a extração da densidade de *bugs* (DB) dos experimentos. A Figura 4 apresenta um gráfico com os dados a DB para cada uma das etapas dos experimentos, para cada participante. Nele, observa-se que a densidade de *bugs* no experimento II é consideravelmente menor em relação a DB do experimento I. Tal característica se deve ao fato de que no experimento II foi necessário apenas alterar os módulos de importação para utilizar o DB2REST, o que implica numa densidade de *bugs* menor — ou inexistente, como foi o caso de P2 —, pois os serviços existentes já foram testados e, portanto, são menos suscetíveis a ocorrência de erros.

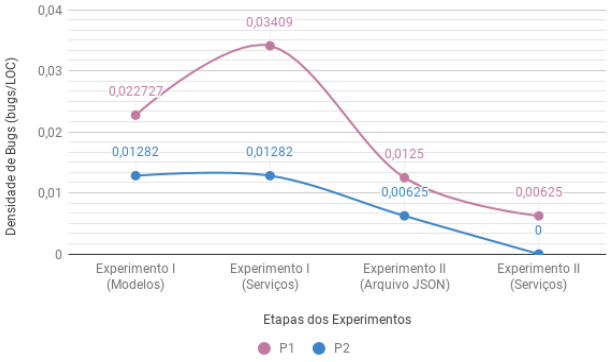

Figura 4. H3: Análise da Densidade de Bugs

De acordo com o gráfico apresentado na Figura 4, e com base no que foi analisado e exposto anteriormente, conclui-se que H3 é verdadeira. A condição $DB_{DB2REST} < DB_{Flask}$ é válida, visto que que utilizar o DB2REST para integrar um *web service RESTful* a uma base de dados legada gera sistemas com menor densidade de *bugs* do que implementar um novo *web service RESTful*.

5. Conclusão

Existem diversas situações em que a integração entre sistemas computacionais é requerida. Entretanto, realizar essa integração perpassa por vários desafios, principalmente considerando-se o uso de sistemas obsoletos, a existência de *hardwares* antigos, falta de documentação adequada dos sistemas, inconsistência de dados etc. Um caso particular desses desafios é a integração entre *web services RESTful* e bases de dados legadas.

Nesse contexto, o presente trabalho propôs a implementação e avaliação de uma solução para facilitar a integração entre esses sistemas. O DB2REST estabelece essa integração através da adaptação entre a camada de serviço e a camada de persistência, de forma que essas interfaces diferentes passam a se comunicar sem a necessidade de modificações estruturais entre eles.

A avaliação da solução proposta foi realizada a partir da análise de dois cenários distintos: no primeiro, os participantes desenvolveram um novo *web service RESTful* em conformidade com a base de dados fornecida, a partir do seu modelo relacional. No segundo, os participantes utilizaram o DB2REST para integrar uma base de dados legada a um *web service RESTful* existente. Observou-se um ganho de produtividade significativo ao realizar a integração entre os sistemas existentes, bem como uma redução da ocorrência de *bugs* em relação ao experimento sem o DB2REST.

Dentre os principais trabalhos futuros previstos para o DB2REST, destacam-se: criação de interface gráfica de usuário para definição do arquivo de mapeamento, avaliação do uso de outras métricas nos experimentos, investigar o uso do DB2REST em outros frameworks para web services e suporte a mecanismos sofisticados de relacionamentos.

O desenvolvimento do presente trabalho possibilitou uma análise sobre como a integração de sistemas computacionais podem trazer resultados satisfatórios. A integração entre um *web services RESTful* e base de dados legada é uma alternativa à criação de novos softwares, medida que viabiliza o reuso de soluções e, consequentemente, a redução nos custos de produção de novos softwares dentro de um mesmo domínio de aplicação.

Referências

Coelho, C. and Sartorelli, R. (2004). Persistência de objetos via mapeamento objeto-relacional. *Bacharelado em Sistemas de Informação, São Paulo*.

Galvão, A. et al. (2009). Alternatives development software, without cost, for micro and small enterprises. *Revista ADMpg Gestão Estratégica*, 2(2):119–123.

Gantan, X. (2014). Python's sqlalchemy vs other orms. Acesso em: Junho 15, 2017.

Lapolli, P. C. (2003). Implantação de sistemas de informações gerenciais em ambientes educacionais. *Dissertação de Mestrado, Florianópolis-SC*.

Pinto, H. L. M. and Braga, J. L. (2005). Sistemas legados e as novas tecnologias: técnicas de integração e estudo de caso. *Informática Pública*, 7:47–69.

Płuciennik-Psota, E. (2012). Object relational interfaces survey. *Studia Informatica*, 33(2A):299–310.