Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Sistemas de Informação

InfraAPI
Um *Plugin* para suporte à integração em ambientes interativos baseados em múltiplas visões

Eduardo Carvalho Ottero e Ribeiro

JUIZ DE FORA
JUNHO, 2017

InfraAPI
Um *Plugin* para suporte à integração em ambientes interativos baseados em múltiplas visões

Eduardo Carvalho Ottero e Ribeiro

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Sistemas de Informação

Orientador: Marco Antônio Pereira Araújo
Co-orientador: Jacimar Fernandes Tavares

JUIZ DE FORA
JUNHO, 2017

InfraAPI
Um *Plugin* para suporte à integração em ambientes interativos baseados em múltiplas visões

Eduardo Carvalho Ottero e Ribeiro

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM SISTEMAS DE INFORMAÇÃO.

Aprovada por:

Marco Antônio Pereira Araújo
D. Sc.

Jacimar Fernandes Tavares
M. Sc.

José Maria Nazar David
D. Sc.

Luiz Felipe Carvalho Mendes
M. Sc.

JUIZ DE FORA

30 DE JUNHO, 2017

Aos meus pais e irmão, pelo amor, apoio incondicional e incentivo

Aos avós, tios e primos, por tornarem a caminhada mais amena

Aos meus amigos, pela força e alegria.

Resumo

[Contexto] Técnicas de integração entre sistemas de software têm sido aplicadas em vários contextos para auxiliar a troca de informações e formar um conjunto de suporte ao desenvolvimento de projetos de computação. Atualmente ainda existem componentes ou etapas em processos de *software*, que não são integrados automaticamente, podendo ocasionar perda de tempo durante a realização de uma atividade e aumento do risco de alterações de dados de *input* e *output*, utilizados durante a troca de informações. De maneira mais específica, foi realizada uma análise na Infraestrutura *GiveMe Infra*, desenvolvida por Tavares (2015), e constatou-se a ausência de um mecanismo automático para extração e importação de dados originados de repositórios de ferramentas que a apoiam. ***[Objetivo]*** Partindo deste cenário, objetivou-se realizar o levantamento de abordagens, técnicas, metodologias e *frameworks* sobre integração de *software* e, com base nesse levantamento, criar um mecanismo chamado *InfraAPI*, hábil para extrair e disponibilizar dados de origem heterogênea, automatizando a comunicação entre componentes ou sistemas. ***[Método]*** Atendendo aos objetivos traçados, foram desenvolvidos *drivers* em uma das camadas da *InfraAPI*, expondo métodos capazes de se conectarem a diferentes repositórios, respeitando a melhor maneira para tal e retornarem resultados conforme o tipo de requisição. Executaram-se testes e uma prova de conceito com dados reais, analisando as vantagens da implementação da solução. ***[Resultados]*** A revisão da literatura retornou resultados satisfatórios para embasar a criação do mecanismo de integração que, por sua vez, gerou resultados de sucesso em sua aplicação, demonstrados pela prova de conceito, garantindo eficiência na proposta de automatização e eficácia em manter a integridade dos dados trocados no processo de integração.

Palavras-chave: Integração, dados, automatização, *GiveMe Infra*, *InfraAPI*.

Agradecimentos

Primeiramente agradeço a Deus por representar uma força superior a qual sempre direciono meu olhar e minhas orações nos momentos de agradecimento e reflexão.

Agradeço também aos meu pais e amigos por nunca desistirem de mim e dos meus sonhos, se hoje sou uma pessoa determinada e persistente é por tudo que absorvi das palavras e diretrizes que sempre me passaram.

Por fim, mas não menos importante, agradeço aos professores Marco Antônio e Jacimar por me conduzirem nos estudos deste trabalho e por me darem a chance de adquirir conhecimento por eles disponibilizados generosamente.

Sem todos vocês eu não teria chegado aqui e não teria condições de continuar.

Tenham minha eterna gratidão.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

DCC	Departamento de Ciência da Computação
UFJF	Universidade Federal de Juiz de Fora
API	*Application Programming Interface*
AIMV	Ambiente Interativo de Múltiplas Visões
RPC	*Remote Procedure Call*
ODBC	*Open DataBase Connectivity*
XML	*eXtensible Markup Language*
RDBMS	*Relational Database Management System*
SQL	*Structured Query Language*
IIS	*Internet Information Services*
DDL	*Data Definition Language*
SOA	*Service-oriented architecture*
SOAP	*Simple Object Access Protocol*
WSDL	*Web Services Description Language*
UDDI	*Universal Description, Discovery and Integration*
QoS	*Quality of Service*
J2EE	*Java2 Platform Enterprise Edition*
JAAS	*Java Authentication and Authorization Service*
JCE	*Java Cryptography Extension*
JSSE	*Java Secure socket Extension*
JDBC	*Java Database Connectivity*
SINAPAD	Sistema Nacional de Processamento de Alto Desempenho
CRUD	*CREATE RETRIEVE UPDATE DELETE*
GUI	*Grafic User Interface*
CGI	*Common Gateway Interface*

1 Introdução

Integração de *software* é tida como a maneira de unir dois ou mais componentes de sistemas para que possam compartilhar seus recursos, sejam dados ou funcionalidades (Martins, 2005). A forma como foi construído cada componente é que vai indicar a melhor técnica de integração a ser aplicada, além das distintas possibilidades, que podem variar entre repositórios de informações, entre estruturas dentro de uma mesma aplicação ou entre aplicações distintas, por exemplo (Martins, 2005). Hamdan e Alramouni (2015) realizaram um estudo apontando que integração de *software* e testes de integração podem representar mais do que 40% do custo global de um projeto.

Na literatura encontram-se diferentes técnicas propostas para realização de integração. Geraldo (2015), por exemplo, discorre sobre uma técnica de integração com sistemas legados conhecida como *Wrapper de tela*, a qual simula o pressionamento de teclas enviando ao sistema legado a sequência correspondente às ações que o sistema externo integrado solicita, como se o usuário tivesse realizado as operações de maneira direta. Outro exemplo de técnica de integração foi abordado por Pang et. al (2015) destacando *web services* como sendo interfaces em forma de serviços que representam funções específicas, consultados via protocolos de *Internet*, independentes da plataforma ou da linguagem em que são construídas cada parte integrada formando uma importante abordagem para intercomunicação entre sistemas, promovendo desenvolvimento de alto nível entre os ambientes de negócios. Já em Rus (2014) foi construída uma proposta de modelo de integração genérica, utilizando componentes *Middleware* ou *Open Database Connectivity (ODBC)*, com o objetivo de serem instalados em um sistema com uma base de dados aqui denominada *"Banco de dados receptor"*, tendo a possibilidade de se conectar às bases de dados de outros sistemas aqui denominadas *"Banco de dados Provedor"*, realizando a troca de informações entre as duas através desta ligação.

Sistemas baseados em diferentes tipos de tecnologias e linguagens de programação são desenvolvidos por vários motivos, tais como a necessidade de reuso de componentes de *software* ou dados já existentes, a integração com sistemas legados, ou mesmo emprego de

linguagens particulares que facilitem o desenvolvimento de diferentes tipos de aplicações (Wileden e Kaplan, 1997).

A *GiveMe Infra* é uma infraestrutura de suporte a tarefas de manutenção e evolução de *software*, desenvolvida por Tavares (2015), integrada ao *Integrated Development Environment (IDE) Eclipse* (Eclipse, 2017) e composta por várias ferramentas interdependentes que se baseiam em conceitos de Ambientes Interativos de Múltiplas Visões *(AIMVs)* (Silva et al., 2012), com variados tipos de abstrações de dados, utilizadas por equipes co-localizadas ou geograficamente distribuídas. Dentre os componentes da *GiveME Infra*, destaca-se o *framework* conceitual *GiveMe Metrics* (Tavares et al, 2014), concebido para auxiliar na decisão para a extração de dados históricos provenientes de repositórios de naturezas variadas, tais como de código fonte, repositórios de defeitos ou repositórios de processos de desenvolvimento em sua utilização. Outra ferramenta importante que compõe a estrutura da *GiveME Infra* é o *plugin GiveMe Views*, concebido para ser apoio na visualização de mudanças e manutenção em projetos de software.

O processo básico de operação realizado na *GiveMe Infra* pode ser segmentado em três partes: (i) extração de dados históricos dos repositórios de defeito de *software*, realizado manualmente pelo usuário (ii) importação dos mesmos dados para a *GiveMe Infra*, também realizado manualmente pelo usuário e, por fim, (iii) visualização das análises automáticas realizadas pelas ferramentas que a compõem. Foi verificado que entre as etapas (i) e (iii), a integração ocorre de maneira semi-automática, pelo fato de depender da intervenção do usuário entre as etapas (i) e (ii). Essa forma de integração, para a operação mencionada, pode ocasionar um certo gasto de tempo e pode expor os dados manipulados a riscos de alterações ou erros de procedimento ao realizar a importação na *GiveMe Infra*.

Baseado no cenário apresentado, concretizou-se o objetivo geral de implementar uma estrutura que realizasse a integração automática entre repositórios e sistemas de *software*, utilizando técnicas que se adéquem aos contextos dos repositórios alvo.

Definiram-se objetivos específicos: (i) levantar (com apoio de um mapeamento sistemático) quais são as técnicas, metodologias, ferramentas, abordagens, *frameworks* ou modelos, existentes na literatura sobre integração de *software* e (ii) mostrar na prática,

a integração entre diferentes sistemas, através do desenvolvimento de um mecanismo automático que permite auxiliar a troca de informações entre a *GiveMe Infra* e diferentes repositórios de dados históricos de defeitos de *software*, visto que atualmente o processo é feito de forma manual. Com isso espera-se otimizar o tempo de execução entre os processos de extração de dados e o processamento pelo *GiveMe Infra*, além de garantir a integridade dos dados que são manipulados entre esses dois pontos de comunicação.

A metodologia de execução do presente trabalho teve início com o levantamento do referencial teórico, que serviu como base de informações fundamentais, apresentando definições aplicadas em seu desenvolvimento. Em seguida, elaborou-se um mapeamento sistemático visando realizar uma pesquisa sobre as técnicas de integração existentes e sob quais formas elas eram abordadas em situações específicas para auxiliar a interligação entre sistemas ou componentes. Foi retornado um conjunto diversificado das técnicas de integração, sob forma de camadas, *frameworks*, ferramentas e abordagens, representando um dos apoios necessários para iniciar o planejamento e execução do mecanismo aqui criado. Abriu-se então uma fase de estudo e aprendizado sobre a estrutura *GiveMe Infra* com a finalidade de entender suas funcionalidades, atributos e limitações. Ao avaliar limitações em conjunto com os resultados do mapeamento sistemático, foi possível identificar o ponto em que pudesse ser elaborada uma solução a ser aplicada em seu contexto, hábil para realizar a troca de dados automatizada entre os repositórios com os quais a infraestrutura se alimenta. Foram utilizados dados armazenados em repositórios de duas ferramentas sobre defeito de *software*, *Bugzilla* (Bugzilla, 2017) e *Redmine* (Redmine, 2017).

Seguindo, realizou-se uma prova de conceito que demonstrou, por testes em cenários previamente estabelecidos, a execução da solução proposta sobre a *GiveMe Infra*, destacando toda a metodologia desenvolvida, bem como as vantagens de sua aplicação.

Este trabalho é organizado em seis capítulos. O primeiro o introduz abordando suas perspectivas, o segundo levanta os pressupostos teóricos como forma de embasamento sobre definições básicas e essenciais para o entendimento do que é discutido aqui. O terceiro capítulo expõe abordagens e formas de aplicação de técnicas de integração entre sistemas em seus diferentes contextos, disponíveis na literatura, retornadas pelo mapeamento sistemático. O quarto capítulo contextualiza a infraestrutura *GiveMe Infra*

(Tavares, 2015), juntamente com a *InfraAPI* a solução propriamente dita, para cumprir com o segundo objetivo aqui proposto, detalhando suas características. No quinto capítulo os resultados produzidos a partir da prova de conceito são detalhados. No último capítulo realizam-se as considerações finais sobre o que este trabalho agregou de maneira a otimizar a utilização da *GiveMe Infra*, suas limitações e trabalhos futuros.

2 Pressupostos Teóricos

Este capítulo irá abordar os pressupostos teóricos referentes às definições fundamentais que compõem este trabalho, para que os assuntos discutidos nos capítulos seguintes sejam melhores compreendidos.

Sistemas de *software* são construídos, geralmente, com o propósito de automatizar uma ou várias etapas de um processo. O estilo de sua arquitetura pode ser conhecido como um modelo convencionado para descrever a maneira de organização do sistema de um domínio específico (Zhao e Zhao, 2010). Os conjuntos de componentes, as regras interativas entre eles, a regra de *design* do protocolo de comunicação, entre outros fatores, são responsáveis por indicar a definição da melhor arquitetura a ser aplicada em sua construção (Zhao e Zhao, 2010). Pode ser composto por diversas camadas, cada uma com seu tipo de abstração e objetivo, previamente definido, entretanto, independentemente desses fatores, a certeza que permeia é a de que o *software* em si utiliza dados como único produto de manipulação.

Os dados manipulados são a representação de fatos, conceitos ou instruções de um modo adequado à comunicação, interpretação ou processamento por seres humanos ou por meios automáticos (IEEE, 1991). Podem ser manipulados de maneiras distintas, e uma dessas maneiras é o seu compartilhamento entre sistemas pois em muitos casos são irreprodutíveis (Mannai e Bugrara, 1993).

O *software* pode ser formado pela junção de várias unidades computáveis, cada uma responsável por oferecer uma funcionalidade, são os chamados componentes, cuja definição dada por Councill e Heineman (2001), os descrevem como sendo elementos que estão em conformidade com um modelo previamente definido para sua construção. Existe a premissa de que ele seja desenvolvido da forma mais independente possível, pelo fato de possibilitar o seu reuso e diversificar sua aplicação, e o que vai definir o quão independente um componente é com relação ao software em que se integra são suas métricas de acoplamento (Yacoub et al., 1999).

Councill e Heineman (2001) também expuseram que um modelo de componente

define padrões específicos de interação e composição do mesmo e, uma implementação de modelo de um componente é um conjunto dedicado de elementos de *software* executáveis necessários para suportar a execução de componentes que estejam em conformidade com o modelo. Uma infraestrutura de componentes de *software* é um conjunto de componentes de interação, projetados para garantir que um sistema ou subsistema construído a partir deles, satisfaçam as especificações de requisitos definidas (Councill e Heineman, 2001).

Dando significado à funcionalidade específica de um módulo ou componente, surge o conceito de coesão, representado pela quantidade de funções distintas que o módulo pode ter. Seu maior grau de coesão é obtido quando se possui função única, ou seja, não executa funções as quais estão fora do escopo de sua ideia de projeto. Em paralelo, surge o conceito de acoplamento, como sendo o grau de interdependência entre pares de módulos. Seu grau mínimo é obtido quando os módulos são construídos de maneira mais independente possível (Offutt et al., 1993). O ideal para a implementação um sistema de *software* então, passa a ser a maximização da coesão e minimização do acoplamento de seus componentes (Offutt et al., 1993).

Foram dadas três razões para tentar alcançar o baixo acoplamento entre módulos (i) quanto menos ligações entre os módulos, menores são as chances de que uma falha em um deles afete o funcionamento de outro (ii) quanto menos ligações entre os módulos, menores são as chances de uma alteração em um deles ocasionar problemas em outros e (iii) quanto menos ligações entre os módulos, melhor fica a compreensão do código (Offutt et al., 1993).

Partindo-se do princípio em que idealiza-se a implementação altamente coesa de um componente, é possível destacar como exemplo o *plugin* de *software*, identificado como um tipo de programa que se integra fortemente com um aplicativo maior para adicionar uma capacidade especial para ele (Mayer et al., 2002). As aplicações maiores devem ser projetadas para aceitar *plugins*, e o fabricante do *software* geralmente publica uma especificação de projeto que permite que as pessoas escrevam *plugins* para ele (Mayer et al., 2002).

Uma importante utilização para *plugins* é na criação de *interfaces* para programação de aplicativo, ou *Application Programming Interface (API)*, conhecida como

o conjunto de símbolos que são exportados e disponíveis para os usuários de uma biblioteca para escrever seus aplicativos (Blanchette, 2008). Mesmo esse conceito parecendo ser genérico, existe uma comum concordância entre os desenvolvedores de bibliotecas sobre estas principais características de uma *API*. Seu *design* é a parte mais crítica pois afeta diretamente no desenvolvimento de aplicações que no futuro se basearão em sua biblioteca(Blanchette, 2008). Uma prática comum para ser aplicada em *APIs* é a implementação e disponibilização de *Web Services*, estruturas que oferecem uma forma padronizada de comunicação entre variados aplicativos de *software* baseadas em troca de mensagens e protocolos bem definidos, podendo ser executados em diferentes plataformas ou *frameworks* (W3C, 2017). São caracterizados por serem capazes de executar operações e entregar os resultados de acordo com seus objetivos e especificações exibidas para sistemas externos no formato *eXtensible Markup Language XML*. Os programas que fornecem serviços simples podem interagir uns com os outros para oferecer serviços sofisticados de valor agregado (W3C, 2017).

O suporte ao armazenamento e manipulação de informações entre componentes ou sistemas pode ser garantido através da utilização de repositórios de dados, que são estruturas desenvolvidas para persistir o estado dos objetos originados na camada da aplicação, de modo a serem recuperados posteriormente (Carey et al., 1995).

Os bancos de dados relacionais são um meio de persistir informações de maneira que fiquem organizadas em tabelas com linhas e colunas (JDBC, 2017). Cada tabela representa um relacionamento comum entre todos os registros que ali estão armazenados (linhas) e, os dados armazenados em uma tabela, podem ser referenciados por chaves de modo a permitir a sua recuperação, por meio de *Structured Query Language (SQL)*(JDBC, 2017). Atualmente, bancos de dados relacionais, como por exemplo *Oracle e IBM DB2*, são utilizados pela maioria dos sistemas de dados, porém ainda existem sistemas que utilizam bancos de dados de outros gêneros (por exemplo, bancos de dados *Information Management System (IMS)*) e sistemas de arquivos (por exemplo, *Virtual Storage Access Method (VSAM)*) (Carey et al., 1995). Esse tipo de banco de dados é de mesma natureza dos repositórios utilizados neste trabalho, por isso deu-se ênfase em sua definição como uma forma de armazenamento e troca de informações.

Todas as estruturas, já descritas até aqui, de alguma forma se relacionam, entretanto é necessário mais do que apenas unir estruturas para construir um sistema com qualidade. Seu ciclo de vida deve ser controlado e medido através da Engenharia de Software, definida como sendo o resultado de uma sequência de abordagens quantificáveis das etapas de desenvolvimento e manutenção de um *software*, suportada por práticas de engenharia ao realizar as atividades como, por exemplo, processos, ferramentas, padrões, organização e gestão (Ansari e Thampi, 2015) ou, segundo Pressman (2011), é uma tecnologia construída e agrupada por camadas cujo o foco na qualidade do *software* desenvolvido é o objetivo principal. As camadas componentes da definição de Pressman (2011) são:

- Processos: servindo como uma ligação entre métodos e ferramentas, possibilitando a visualização e desenvolvimento das etapas de criação de *software*.
- Métodos: são as abordagens detalhadas de como se desenvolver o *software*.
- Ferramentas: meio pelo qual se realiza o desenvolvimento.

O termo "qualidade" supracitado, refere-se aos atributos que devem ser considerados de modo a garantir a fidelidade aos requisitos para construção do *software*, tais como: conteúdo funcional, taxas de erro, *performance*, usabilidade, dentre outras (Humphrey, 1988). O nível de maturidade de uma aplicação desenvolvida depende de um processo de desenvolvimento bem estruturado e regido por metodologias formais, baseadas em atividades vitais para converter os requisitos de usuários em *software*. Esse processo pode incluir, conforme necessário, especificações de requisitos de usuário, desenvolvimento, implementação, verificação, instalação, suporte operacional e documentação, além das eventuais tarefas de manutenção e melhorias (Humphrey, 1988).

A organização ou união das práticas e conceitos aqui já definidos, não necessariamente todos, convergem em um produto de *software* que passa por versões e incrementos durante seu ciclo de vida. Em um dado momento de seu ciclo pode surgir a necessidade de obter informações de fontes distintas que não estejam no domínio original do produto, porém estão prontas e disponíveis em outros domínios. Surge assim a pratica da implementação de uma abordagem de integração automatizada destes produtos inseridos em diferentes domínios.

O desenvolvimento de aplicativos está buscando cada vez mais utilizar a integração entre múltiplas estruturas (por exemplo, *Graphical User Interface (GUI)*, sistemas de comunicação, bancos de dados, entre outros) juntamente com bibliotecas de classes, sistemas legados e componentes existentes. Porém existe um impeditivo de que muitas estruturas de versões passadas foram projetadas para extensão interna, ao invés da comunicação externa com outros *frameworks*, isso tem sido observado como uma grande preocupação na tecnologia da informação (Ansari e Thampi, 2015).

O conceito de integração de *software* surge, segundo Barrett et al. (1996), como o processo pelo qual múltiplos módulos de *software* (programas, subprogramas, conjuntos de subprogramas, entre outros) são feitos para operar em conjunto trocando dados entre si.

As abordagens de integração podem ser classificadas como "soltas", em que os módulos paticipantes têm baixo acoplamento entre si, ou "apertadas", em que os módulos possuem alto acoplamento entre si (Barrett et al., 1996). A integração solta reduz o impacto em um sistema quando módulos são adicionados ou alterados. A integração apertada exige um maior grau de alteração em um sistema, quando módulos são adicionados ou alterados (Barrett et al., 1996).

Sob o ponto de vista de comunicação entre módulos, existem duas principais maneiras: ponto-a-ponto e *multicast*. A comunicação ponto-a-ponto aborda que os dados são trocados diretamente de um módulo de *software* para outro (Barrett et al., 1996). Duas abordagens ponto-a-ponto comuns são a chamada de procedimento e a *application-to-application*. A chamada de procedimento envia parâmetros de procedimento de um módulo de *software* conhecido como o chamador para outro conhecido como a chamada, retornando opcionalmente valores para o chamador. Na abordagem de *application-to-application* os programas de aplicativos têm *IDs* exclusivos e outros programas enviam mensagens para eles usando os *IDs* como referências de endereços (Barrett et al., 1996). A comunicação *Multicast* significa que os dados são enviados de um módulo de *software* para um conjunto de outros módulos. Duas abordagens *multicasting* populares são a invocação implícita e o barramento de *software*. Na abordagem de invocação implícita, os módulos de *software* expressam seu interesse em receber certos tipos de dados que

são encaminhados, normalmente por um servidor, para os destinatários apropriados. Na abordagem de barramento de *software*, os módulos de *software* têm suas entradas e saídas ligadas aos canais de um barramento abstrato. Os dados enviados para um canal de barramento são recebidos por todas as ferramentas com uma entrada ligada a esse canal (Barrett et al., 1996).

Ao manipular dados, deve-se ser capaz de perceber quais são seus atributos, origens, destinos e compatibilidade com o componente que está manipulando. Davis et al. (2000) levantam a discussão de que problemas surgem quando sistemas complexos são construídos através da integração de componentes distintos, frequentemente heterogêneos. Ao analisar o *design* da arquitetura do sistema e seus componentes, incompatibilidades potenciais podem ser previstas no momento em que se decide realizar a troca de informações com outros sistemas.

A a aplicação de técnicas de integração são o desígnio de estudo do presente trabalho. O capítulo que se segue detalha o mapeamento sistemático realizado, representando o primeiro objetivo específico deste trabalho.

3 Mapeamento Sistemático

O mapeamento sistemático é uma metodologia que fornece subsídios para que o pesquisador apresente sua abordagem utilizada ao realizar sua pesquisa, considerando alguns objetos de apoio como o volume de publicações sobre o assunto em foco, bem como os tipos de pesquisa (Steinmacher e Gerosa, 2012). O protocolo de mapeamento sistemático, segundo Kitchenham (2004), é a formalização da sequência metodológica utilizada pelo pesquisador para exprimir suas buscas sobre o assunto analisado.

3.1 Planejamento

O protocolo planejado constitui-se de objetivo, critérios para inclusão, critérios para exclusão, questões de pesquisa, *string* de busca e bases de dados de publicações. Seguem detalhadas as etapas.

3.1.1 Objetivo

Verificar o estado da arte relacionado à automatização da troca de informações entre sistemas, através de técnicas de integração, alçando a melhoria de processos e integridade dessas informações trocadas.

Busca-se este entendimento através da literatura existente e quais são as abordagens e metodologias utilizadas para tratar a questão, pelos conhecimentos já adquiridos, ou pelas abordagens que ainda estão em desenvolvimento.

3.1.2 Critérios para Inclusão (CI)

Os critérios para esta etapa foram estabelecidos para guiar a seleção de publicações existentes, aderentes à proposta de se criar uma solução integrada para automatizar o processo de leitura e importação de dados para análises, em formatos específicos.

Os critérios são:

- (CI): Capacidade de extração e manipulação de dados de fontes heterogêneas, por meio de metodologias de integração, tais como chamadas de métodos de *API*, acesso direto por banco de dados, entre outros.

- (CI): Capacidade de inclusão ou disponibilização de dados manipulados para consumo por outros sistemas/componentes, por meio de metodologias de integração.

3.1.3 Critérios para Exclusão (CE)

Os critérios para esta etapa foram definidos com a finalidade de que as abordagens que não atendam ao foco da execução da busca, não sejam incluídas na pesquisa.

Os critérios são:

- (CE): Capítulos de livros, chamada para congressos.

- (CE): Pesquisas que não podem ser acessadas por completo.

- (CE): Pesquisas que não apresentam técnicas, metodologias, ferramentas, abordagens, *frameworks* ou modelos que descrevam como realizar integração entre sistemas ou repositórios de dados.

3.1.4 Questões de Pesquisa

A partir do objetivo e dos critérios para avaliação das propostas de mecanismos de integração, foram definidas as questões de pesquisa, com a finalidade de manter o contexto da execução do mapeamento sistemático. Abaixo encontram-se as questões:

- (Q1): Quais tecnologias (técnicas, metodologias, ferramentas, abordagens, modelos ou *frameworks*) já foram obtidas, relacionando integração entre repositórios ou sistemas? O objetivo desta questão é buscar por estudos que abordam a fase de integração, que é um dos passos para que haja a automatização da troca de informações entre sistemas.

- (Q2): Quais das abordagens que permitem a integração de dados entre diferentes tecnologias (*softwares*, por exemplo) são automáticas ou semi-automáticas, isto é,

sem interação ou com interação parcial por parte do usuário? O objetivo desta questão é identificar como as abordagens semi-automáticas ou automáticas são estruturadas e se atendem ao objetivo desta pesquisa.

3.1.5 *String* de Busca

Para atender aos critérios de inclusão, exclusão e questões de pesquisa, definiu-se a seguinte *string* de busca:

- ((*technique* OR *methodology* OR *tool* OR *approach* OR *framework* OR *model*) AND (*integration* OR *interoperability*) AND (*repository* OR *repositories*) AND (*database*)).

3.1.6 Bases de Dados de Publicações

A execução da *string* de busca foi realizada em três bases de dados que contêm publicações eletrônicas e são disponibilizadas de maneira livre para acadêmicos da Universidade Federal de Juiz de Fora (UFJF). Foram utilizadas as bases:

- *ACM Digital Library (http://portal.acm.org).*
- *IEEE Digital Library (http://ieeexplore.ieee.org).*
- *Science@Direct (http://www.sciencedirect.com).*

A definição do protocolo do mapeamento sistemático realizada oferece uma análise formal das técnicas buscadas, em contrapartida a uma pesquisa *ad-hoc* que não garante uma sistematização mais clara para se obter retorno baseado em investigação(Kitchenham, 2004).

3.2 Execução do Mapeamento Sistemático

A *string* de busca foi executada nas bases de publicações supracitadas aplicada aos metadados dos artigos (título, resumo e palavras-chave), em conformidade com o que foi definido nos critérios e questões e objetivos do protocolo do mapeamento sistemático. O

resultado foi analisado e validado. As bases de publicações em que foram executadas a *string* foram consideradas suficientes pelo fato de disponibilizarem seus conteúdos de maneira íntegra e gratuita para os alunos da UFJF, além do fato de tratarem de assuntos científicos coerentes com as propostas de pesquisa aqui estabelecidas.

A execução da *string* foi realizada em março de 2017.

Para o controle e análise de métricas sobre artigos retornados na execução da *string* de busca, foi utilizada a ferramenta *StArt* (StArt, 2017). Esta ferramenta tem a capacidade de ler e importar dados no formato *bibTex*, e garantir todas as funcionalidades referentes ao gerenciamento dos resultados de pesquisa, como remoção de arquivos duplicados, análises das metodologias aplicadas através de gráficos e o controle e armazenamento dos arquivos que foram importados no formato *bibTex*.

A análise inicial automatizada da ferramenta evidenciou que: entre os 523 artigos retornados na execução da *string* de busca nas bases, 26 foram considerados duplicados. Esses foram confirmados como tal pelo desenvolvedor desse trabalho.

Considerando a possibilidade de ainda haver artigos duplicados por conta de diferenças mínimas em seus títulos, autores, datas de publicação e outros atributos, foi necessária uma segunda verificação manual mais detalhada após a análise automatizada pela ferramenta. Nesta segunda investigação foram detectadas 19 publicações ainda duplicadas.

Com base nas 478 publicações restantes, foi iniciada a verificação dos resumos destas publicações, das quais 73 foram aceitas e, posteriormente à leitura integral dos textos, 12 foram satisfatórias ao objetivo da pesquisa.

Na fase final da execução do mapeamento sistemático, incluiu-se o artigo de Geraldo (2015) pelo fato de não constar em nenhuma base utilizada mas, mesmo assim, foi relevante por abordar conceitos de técnicas de integração que foram utilizados no desenvolvimento da solução proposta.

A análise sobre esse trabalho e os estudos escolhidos como aderentes à proposta de pesquisa são apresentados a seguir.

3.3 Resultados do Mapeamento Sistemático

Recentemente Geraldo (2015) realizou um levantamento expondo algumas técnicas utilizadas em integração entre sistemas legados simulando cada uma delas para demonstrar seus comportamentos. As técnicas levantadas foram *Wrapper* de tela, simulando o pressionamento de botões através de envio de comandos ao sistema legado , *Wrapper* de banco de dados, permitindo a comunicação direta entre aplicações externas e o banco de dados do sistema legado, *Wrapper* de código fonte, realizando a reutilização de rotinas criadas no código fonte do sistema legado através de *interfaces* e, por fim, *web services*, que são aplicações que fornecem serviços a aplicações clientes como se fossem servidores, independente da plataforma em que a aplicação foi construída. A conclusão deste estudo veio para destacar que cada uma das técnicas expostas tem vantagens e desvantagens e, portanto, deve-se avaliar cada necessidade específica de integração para se escolher a implementação mais adequada. Em estudo anterior, seguindo pelo caminho do acesso a dados por meio de camadas e estruturas que recuperam informações diretamente no repositório a ser integrado, Draper et al. (2001) desenvolveram uma ferramenta chamada *Nimble* que consegue promover uma *interface* com variadas fontes de dados, podendo ser bancos de dados modernos, sistemas legados ou arquivos estruturados. Essas *interfaces* serão a via para que ocorra a troca de informações, utilizando modelo de dados em *XML* como característica chave do produto desenvolvido. Destacaram que as vantagens de se utilizar *XML* como modelo residem no fato de sua flexibilidade e assimilação mais atraente para o usuário. A arquitetura do sistema é formada pela sua camada *Front End*, camada de integração com um servidor de metadados e as fontes de dados representadas pelo modelo de dados *XML*

Os usuários e aplicações interagem com o *Nimble* utilizando uma série de esquemas mediados que são representações das visões das fontes de dados. Quando a consulta é colocada ao mecanismo de integração ela é analisada e quebrada em múltiplos fragmentos baseados nas fontes de dados alvo. O compilador traduz cada fragmento na linguagem de consulta apropriada à fonte de destino. Após esta etapa, o servidor de metadados processa a consulta pois tem os mapeamentos que permitem a consulta ser traduzida e realizada na respectiva fonte de dados. Por fim, o retorno é exibido no *Front End* para o

usuário. Sua arquitetura é representada na figura 3.1.

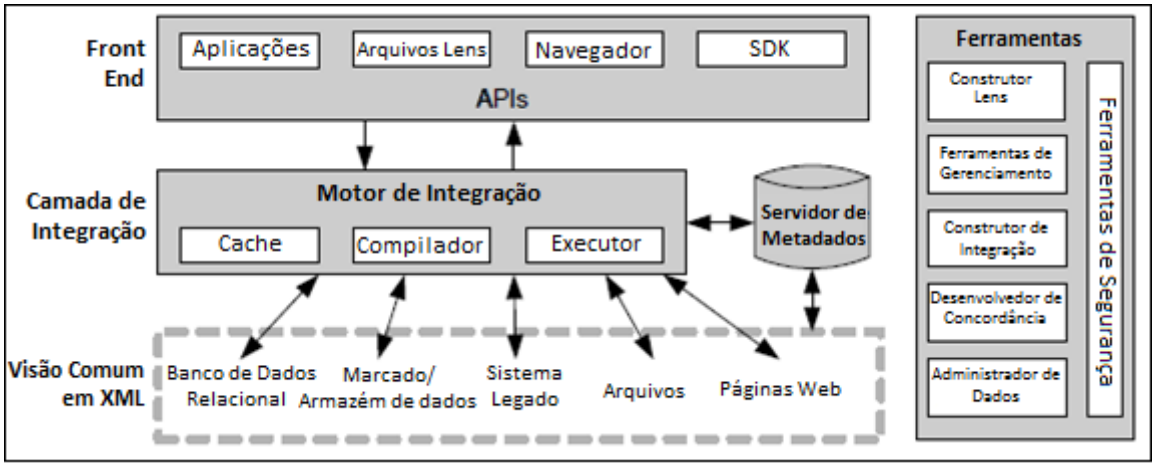


Figura 3.1: Arquitetura *Nimble*

De forma semelhante a Draper et al. (2001), Naidu et al. (2007) apresentaram uma abordagem para integração de dados em tempo de execução ou *on-the-fly*, com o objetivo de combinar o fluxo efetivo de informações entre diferentes fontes de dados e exibi-las em um *framework* integrado para o usuário final. Consiste na implementação de um sistema de consultas, centrado no usuário, que busca as informações dessas consultas, na *internet*, a partir de múltiplas fontes biológicas e organiza uma variedade de informações dentro de uma visão unificada homogênea, chamado *BioXBase*. Eles utilizam uma arquitetura com um mediador *three-tier* (três níveis distintos) de integração para nomear de maneira semântica os bancos de dados heterogêneos, o módulo de compilação, o módulo de execução e a camada de apresentação. A ideia é que, quando um usuário realiza uma consulta no mediador, ela seja transformada em uma outra apropriada sobre a fonte de dados pertinente à necessidade do usuário, sendo cada módulo da arquitetura responsável por uma etapa de manipulação, construção e exibição dos resultados da consulta. Conseguiu-se, desta forma, agregar um volume de informação e inteligência maior às requisições dos usuários pelo fato desta integração alcançar um grande número de repositórios, inclusive pela possibilidade de cadastrar novos repositórios a serem atendidos pelo *BioXBase*.

A questão da necessidade do reaproveitamento de dados para realizações para compartilhamento não é recente, na década de 90, Mannai e Bugrara (1993) discutiram sobre o fato de que existem milhares de repositórios médicos dispersos que guardam

informações consideradas muito valiosas, isso faz com que seja altamente desejável o compartilhamento e facilitação de acesso a estas informações pelos membros da comunidade médica e afins. Para atender a esta necessidade, criaram uma plataforma comum para troca de dados entre vários bancos de dados médicos, representada pela figura 3.2.

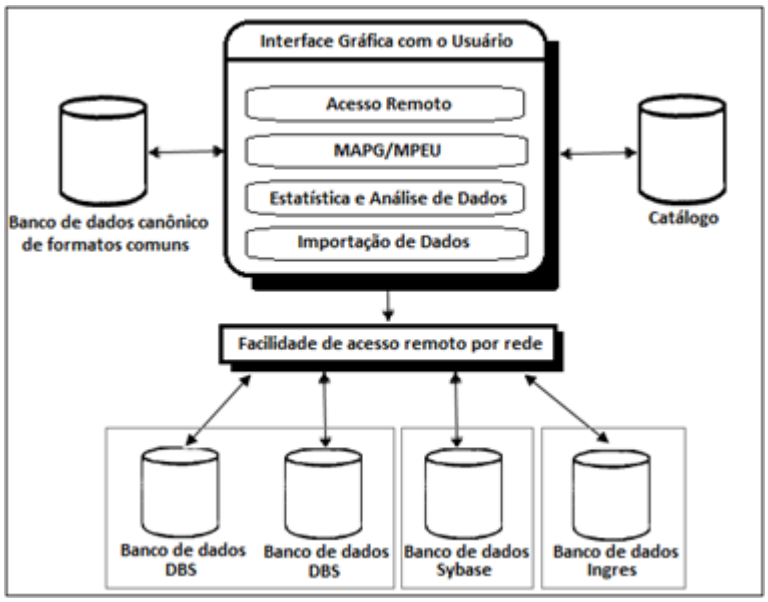


Figura 3.2: Banco de dados, Repositório de Informações

A estrutura implementa as capacidades de acesso por uma única interface gráfica a diversos bancos de dados heterogêneos, além de ser composta por ferramentas de importação de dados, gerador de aplicações de múltiplos bancos de dados, interfaces de pacotes estatísticos e controlador de execuções de aplicações de múltiplos bancos de dados. Através de um dos componentes da interface gráfica do usuário, o Gerador de Programas de Acesso a Múltiplos Bancos de Dados *(MAPG)*, pode-se escrever programas para acessar as múltiplas bases heterogêneas indicando as necessidades de informações e as bases que serão acessadas para recuperá-las. As aplicações geradas pela utilização do *MAPG* podem ser incluídas na fila de processamento da Unidade de Execução de Programas de Múltiplos Bancos de Dados *(MPEU)*, responsável por executar o programa e retornar o resultado para o usuário.

Ainda na década de 90, Thiran et al. (1998) descrevem uma arquitetura generalizada, uma metodologia e um ambiente de *Computer-Aided Software Engineering (CASE)*, destinados a proporcionar a usuários e programadores uma interface abstrata para bancos de dados independentes, heterogêneos e distribuídos.

Para integrar os bancos de dados foi utilizada uma abordagem federada chamada *InterDB*. De acordo com esta abordagem, cada banco de dado local é descrito pelo seu próprio esquema a partir do qual um subconjunto, chamado esquema de exportação, é extraído e, logo em seguida, é mesclado no esquema global. Cada aplicação interage com o dado local através de uma visão, derivada do esquema global. O processo de integração se baseia em três passos: recuperação de esquema conceitual em que são manipulados os esquemas a fim de recuperar apenas suas construções explícitas; integração conceitual em que as igualdades semânticas das diferentes fontes e dados são comparadas e unidas em forma de tipo e subtipo quando possuem o mesmo significado de seus registros; definição de mapeamento, responsável por interpretar e converter a consulta para o esquema global em consultas para os esquemas locais, baseado nas construções feitas pelos passos anteriores. A figura 3.3 representa os passos do InterDB.

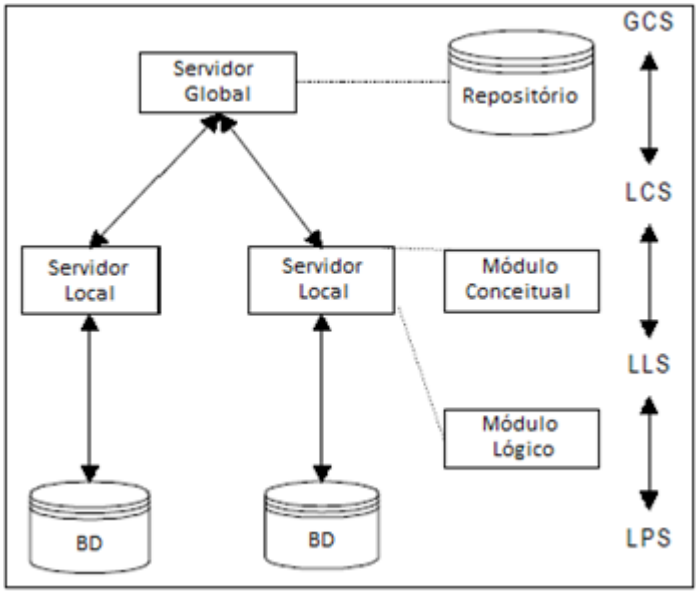


Figura 3.3: Arquitetura de um banco de dados Federado *InterDB*

A arquitetura do *InterDB* é composta por uma hierarquia de camadas mediadoras, chamadas de servidores locais, dedicados a cada banco de dados local e um servidor global, dedicado ao esquema global conceitual. Todos os mediadores suportam as operações que o *InterDB* realiza transformando os dados reais em uma base de dados homogênea virtual, dinamicamente, além de realizar a operação reversa de transformar as consultas na base de dados global em consultas nas bases locais.

Em outro estudo Marinos et al. (1991) propuseram uma arquitetura que con-

sistia em uma camada intermediária e unificadora entre aplicações e bancos de dados heterogêneos para realizar integração em um único conglomerado lógico. Pode ser vista como um mecanismo que tem a capacidade de distinguir e combinar dados de diferentes fontes, gerando informações relevantes para uma determinada aplicação cliente. É capaz de armazenar diferenças semânticas e funcionais de variadas aplicações, através da interpretação de suas requisições, para abstrair as representações locais de dados, além da capacidade de realizar tradução de diversas terminologias de consultas diferentes. Posteriormente, realiza a homogeneização de dados heterogêneos e os retorna para a aplicação cliente. Sua representação é dada pela figura 3.4.

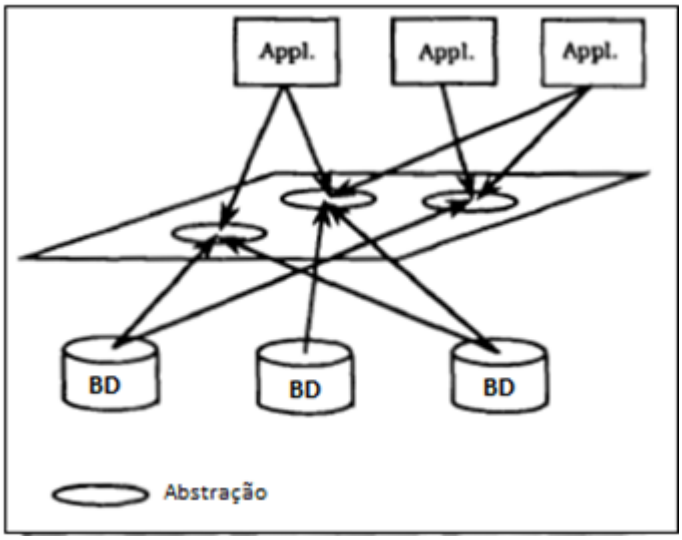


Figura 3.4: Um mediador entre bancos de dados e aplicações (Marinos et al., 1991)

Eles se basearam e detalharam as possíveis abordagens existentes, tais como Abordagem Logica Centralizada, Arquiteturas Federadas, Arquitetura de *cluster*, para propor a solução mediadora genérica.

De maneira mais específica,Lee e Jeong (2006) apresentaram a implementação de um *framework* capaz de integrar e trocar dados entre diferentes fases do ciclo de vida nos projetos de construção de pontes de aço. A aplicação deste *framework* se baseia em um modelo arquitetônico de produto previamente definido. Sua estrutura geral é dada pela figura 3.5.

Para realizar o mapeamento de dados entre dois modelos distintos, utiliza-se um banco de dados relacional centralizado, com toda sua estrutura de gerenciamento *(RDBMS, ODBC e SQL)*, além do suporte a *web services* pelo *Internet Information Ser-*

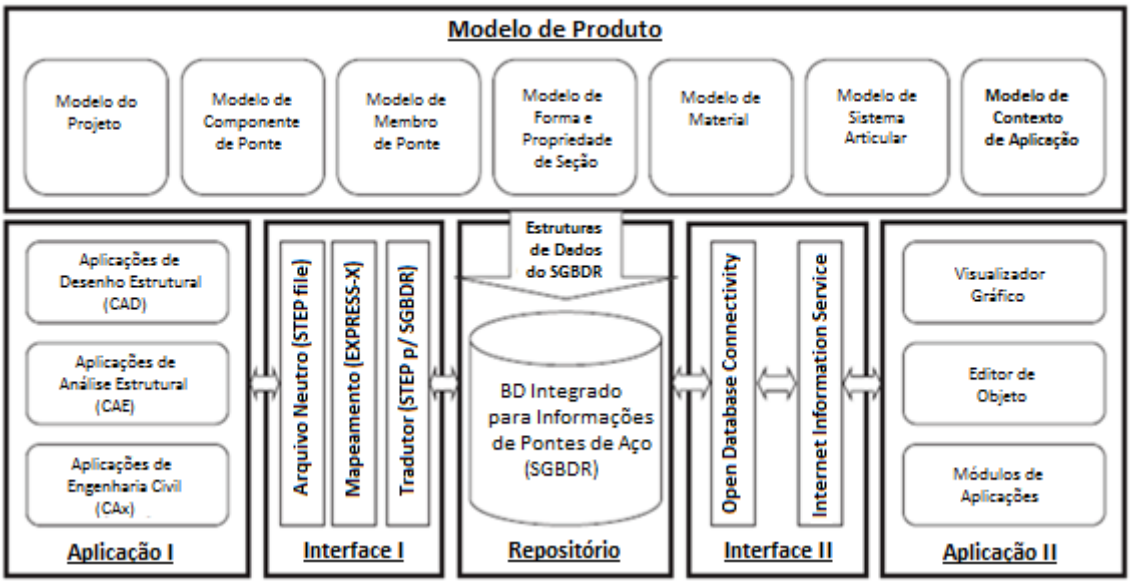


Figura 3.5: Um *Framework* para gerenciamento de informações de pontes de aço

vice (IIS) permitindo o acesso controlado pelas aplicações através da rede. As *interfaces* entre as aplicações e o banco de dados são responsáveis por traduzir arquivos físicos no formato *Standard for the Exchange of Product model data (STEP)*, um tipo de formato utilizado por aplicações de projeto e planejamento de estruturas de engenharia, realizar a conversão para linguagem de definição de dados *(DDL)* inserindo estes dados no repositório e, ao mesmo tempo, podem promover informações contidas no repositório para outras aplicações, também no formato *STEP*.

Olhando sob outro ponto de vista dentre as técnicas de integração pesquisadas, Borkar e Vinayak (2004) apresentaram a *Liquid Data for WebLogic*, uma ferramenta da *BEA Systems* que possui uma abordagem baseada em *web services*, *XQuery* e esquemas *XML*, para promover acesso integrado a fontes de dados heterogêneas. Ela foi projetada para ter uma estrutura que produz e publica serviços que consomem dados de diversas fontes e serviços. Em suma, a *Liquid Data* consegue disponibilizar as fontes de dados, para o desenvolvedor, em formato *XML*, independente o seu tipo original, permitindo a criação de funções *XQuery* para se obter o dado desejado. Sua arquitetura é representada pela figura 3.6.

No núcleo da *Liquid Data* há um motor que é responsável por compilar e executar as consultas *Xquery* nas diferentes fontes de dados e serviços, de acordo com a consulta criada em um editor de *Xquery*, chamado *Data View Designer*, outro componente da

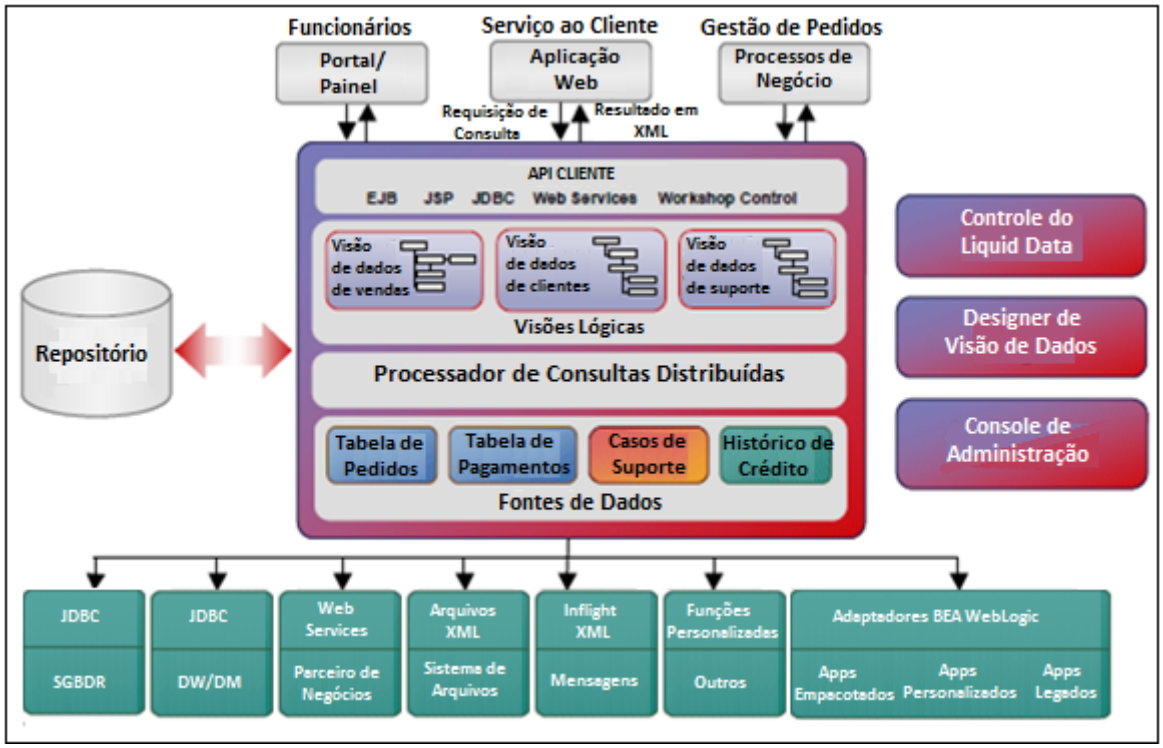


Figura 3.6: *Liquid Data for Weblogic* da *BEA*

ferramenta, que permite a navegação entre as fontes de dados e serviços e as funções *XQuery* de maneira gráfica. Uma *API* também é oferecida na ferramenta com o objetivo de facilitar o desenvolvimento de aplicações clientes.

Outra proposta para realizar comunicação automatizada entre *softwares* foi apresentada por Seligman et al. (2010) que, motivados pelo fato de ferramentas de integração já desenvolvidas cobrirem uma pequena parte das deficiências de troca de informações, desenvolveram o *OpenII*, uma plataforma escalável e expansível, bem como uma série de ferramentas de integração de informação de força industrial construídas sobre esta.

Dentre seus componentes pode-se destacar o repositório compartilhado *Yggdrasil*, que implementa um metamodelo (chamado *M3*) neutro e estendido para ambos os esquemas e mapeamentos. Possui também uma variedade de importadores e exportadores incluindo esquema *XML*, *SQL DDL*, *Web Ontology Language (OWL)* e planilhas *Microsoft Excel*, e ferramentas de componentes interoperando pelo compartilhamento de conhecimento através dos repositórios via *web services Java* ou outras *APIs* específicas para certos tipos de ferramentas (como casamento de esquemas). As várias ferramentas que abordam desafios específicos de integração de informação tais como: busca de esquema *Schemr*, que ajuda usuários a descobrir e visualizar esquemas relevantes no *Ygg-*

drasil, incentivando compartilhamento e reuso; o Agrupamento de esquema e visualização *Affinity*, que promove uma clara visualização dos ativos de informação e oportunidades para compartilhamento de dados; *Harmony*, que realiza um processamento linguístico para identificar o casamento entre esquemas; *RMap* e *XMap* que realizam a geração de código de troca de dados para *SQL* e *XQuery* respectivamente. Interpretam um conjunto de correspondências como uma coleção de dependências de geração de tuplas. Compartilham uma representação comum de mapeamentos entre si e, por isto, os conhecimentos acumulados podem ser aproveitados de forma mútua. É possível visualizar toda a estrutura implementada da ferramenta através do esquema da figura 3.7.

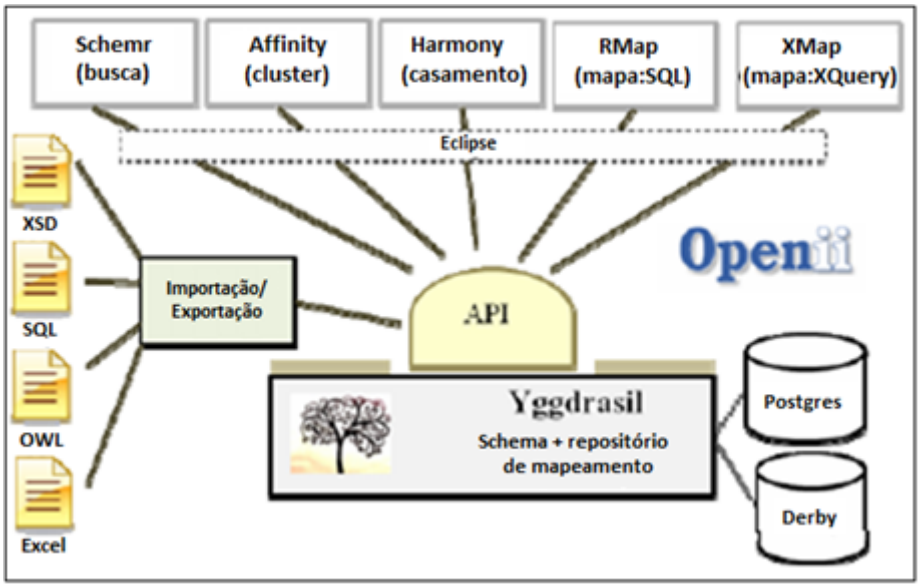


Figura 3.7: *OpenII*: Arquitetura e Ferramentas Componentes

Yong-Gang Gong e Xin Chen (2010) também utilizaram uma arquitetura orientada a serviços *(SOA)* como solução para os desafios da captação de dados em fontes médicas heterogêneas.Implementaram uma plataforma chamada Plataforma compartilhada para integração de informação de cuidados com saúde *(HIISP)*. Inicialmente descreveram a abordagem que leva os princípios da arquitetura orientada a serviços, detalhando suas camadas e funções, que são: *Simple Object Access Protocol (SOAP)* como camada de transporte de mensagens entre o consumidor do serviço e o provedor, *Web Services Description Language (WSDL)* como descritor do serviço, *Universal Description Discovery and Integration (UDDI)* para registrar e visualizar os serviços disponibilizados, *Web Services Interoperability (WS-I)* como camada de interoperabilidade e testes dos serviços,

plataforma de desenvolvimento para aplicações *Service-Oriented Architecture (SOA)* que fornecem um *framework* para os desenvolvedores criarem e publicarem os serviços e garantindo sua escalabilidade e *performance*, *Quality of Services (QoS)* para fornecer todos os requisitos de sustentabilidade da plataforma tais como gerenciamentos dos serviços, segurança, coordenação, entre outros.

Logo em seguida apresentaram especificamente os componentes que formam a arquitetura do *HIISP*. Constituída por quatro componentes principais, o primeiro é um repositório de dados normalizado com base nos padrões Saúde Nível Sete *(HL7)*. O segundo é formado por um conjunto de serviços de gerenciamento de dados responsável pela coerência dos dados exibidos para as organizações de saúde de modo que haja uma única visão do paciente e fornecedores. O terceiro é uma plataforma de mensagens que permite a comunicação entre diferentes sistemas de informação. O quarto é uma plataforma de desenvolvimento compatível com *Java 2 Platform Enterprise Edition (J2EE)* que dá suporte à criação de aplicações de saúde independentes.

Em outra publicação, Goodall et al. (2008) propuseram uma abordagem para integração de informações de fontes de dados distintas pela criação de uma camada intermediando o banco de dados do *National Water Information System (NWIS)* e os sistemas que pretendem acessá-lo. A arquitetura foi desenvolvida baseada em *SOA* para atingir o objetivo de troca de informações.

A aplicação cliente realiza requisições *SOAP* no servidor, baseado no arquivo *WSDL*. Logo em seguida, o servidor interpreta a requisição e responde com o retorno necessário.

Os serviços desenvolvidos para o *NWIS* envolvem o processo de captura da tela de construção da *Uniform Resource Locator (URL)* e análise da página *web* de retorno, o que quer dizer que eles não têm acesso ao banco de dados da *NWIS* e isto implica em limitações na capacidade de realizar consultas personalizadas no mesmo, porém pode-se implementar um algoritmo de busca para realizar a leitura de metadados locais, com isso os serviços de entrega de dados seriam capazes de manter estes metadados sempre atualizados, sem necessidade de realizar a pesquisa no banco de dados remoto.

As publicações sobre integração por meio de *web services* na construção da auto-

matização da troca de informações têm destacado a maior vantagem deste tipo de técnica, que é a independência entre plataformas ou componentes que estão se comunicando, uma vez que todo o processo de comunicação é realizado por uma linguagem comum em uma camada intermediária a estas estruturas. Além disso, é possível unir este tipo de implementação com outras técnicas, assim como Masseroli et al. (2004) solucionaram a questão da recuperação, integração e avaliação de dados heterogêneos e imagens da área médica, utilizando duas das técnicas expostas por Geraldo (2015). Implementaram uma arquitetura cliente-servidor baseada em agentes de *software*, exibida na figura 3.8.

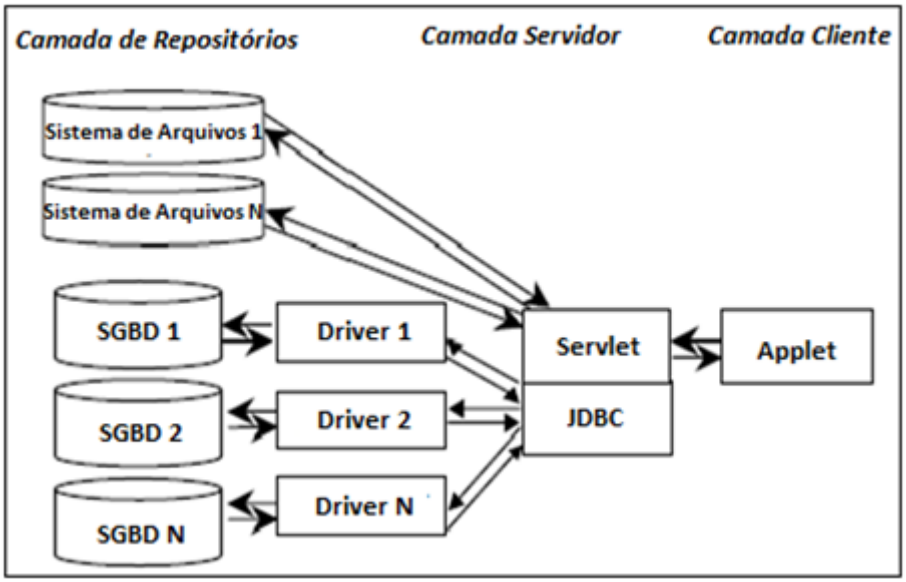


Figura 3.8: Arquitetura Cliente Servidor em três camadas de comunicação entre um *Applet* e os dados médicos armazenados em repositórios distribuídos.

O lado cliente compreende um *applet Java* com uma *interface* para o usuário navegar e visualizar diferentes dados de exames médicos e de pacientes, integrando-os. O lado servidor gerencia conexões seguras e consultas para bancos de dados heterogêneos remotos e sistemas de arquivos, contendo dados clínicos e pessoais de pacientes. Esta arquitetura está disposta em 3 camadas, a primeira é formada por diferentes repositórios de dados em diferentes plataformas, incluindo diferentes bancos de dados e sistemas de arquivos; A camada intermediária é constituída de um servidor *web* e um *servlet Java*, implementando serviços de acesso aos repositório; A terceira camada é representado por um *applet Java* instalado em um navegador *web* no cliente quando este se conecta ao servidor *web* na camada intermediária.

São utilizadas invocações remotas de métodos para comunicação com o *servlet*

na camada intermediária que, por sua vez, garante uma conexão segura e suporte às consultas, através da utilização de várias *APIs* de segurança *Java - Java Authentication and Authorization Service (JAAS), Java Cryptography Extension (JCE) e Java Secure socket Extension (JSSE)* - e a *API Java Database Connectivity (JDBC)* como interface para bancos de dados específicos do fornecedor.

Os agentes de *software* foram desenvolvidos para realizarem uma análise automática das consultas, no lado servidor, e identificarem o formato apropriado para exibirem os dados no lado cliente.

De forma semelhante à publicação de Masseroli et al. (2004), Orzechowski e Dziech (2011) propuseram um sistema com o objetivo de possibilitar uma melhor eficiência e controle no processamento de dados policiais originados de sistemas e fontes de dados heterogêneos. Esta proposta é formada por uma federação de repositórios de dados, um sistema intermediador *(Brokerage system)*, responsável pelo processamento de informações e gerenciamento da federação dos repositórios, e um módulo de recomendação de nível superior, que é o núcleo do sistema. A comunicação entre os componentes ocorre por *web services* disponibilizados nos sites de cada repositório.

Para que a camada do sistema intermediador tivesse uma execução efetiva, o autor utilizou-se da ferramenta *open source Apache ActiveMQ* para comunicação remota e provedora de padrões de integração. Foi utilizado, também, o *framework Apache Camel* integrado ao *ActiveMQ* de modo a implementar regras de encaminhamento e mediação no ambiente de maneira inteligente. Desta forma, tornou-se possível a organização desta camada em níveis Produtor, *ActiveMQ + Camel* e Consumidor trabalhando a troca de mensagens sob o mecanismo de filas.

Desenvolveu-se um protótipo que permite a troca de mensagens por uma *interface* gráfica simples e permite que os *web services* publicados na camada intermediadora fossem capazes de executar busca em cada banco de dados e criar a federação de repositórios, representado pela figura 3.9.

O mais importante a se ressaltar em todos as publicações selecionadas é a constante presença de uma arquitetura centrada em uma camada que realizará toda a unificação e controle de dados de origens distintas para fornecer a informação composta,

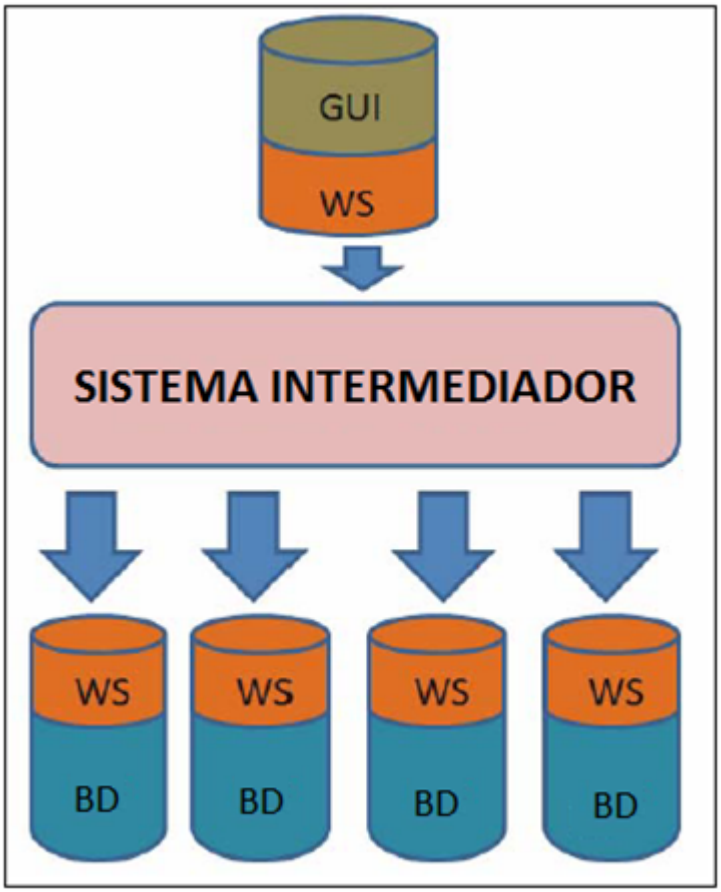


Figura 3.9: Protótipo do Sistema Intermediador

conforme requerida, de maneira transparente, isto é, sem que haja preocupações com a origem daquela informação por parte de usuários ou aplicações.

É válido destacar, também, que as aplicações para as técnicas de integração existentes se apresentaram muito flexíveis com relação aos contextos em que foram aplicadas, mostrando, assim, a infinidade de maneiras de trabalho e seus grandes impactos e importância para o desenvolvimento de uma sistemática de troca de informações e dados mais coesa, precisa e reutilizável.

A tabela 3.1 exibe um comparativo entre as percepções de acoplamento, percepções de coesão, número de técnicas de integração utilizadas, unificação semântica dos dados e possibilidade de utilização como *plugin*. Os critérios utilizados na tabela foram estabelecidos de acordo com a necessidade de identificar, dentre os estudos, quais se relacionam ao que se almeja atingir como solução de integração: baixa percepção de acoplamento, alta percepção de coesão, aplicação de mais de uma técnica de integração, sem unificação semântica e utilização como *plugin* para o Eclipse.

Abordagem Proposta	**Percepção de acoplamento com os componentes integrados**	**Percepção de coesão**	**Número de técnicas de integração distintas utilizadas**	**Realiza unificação semântica entre dados**	**Pode ser utilizado como plugin do Eclipse**
Geraldo (2015)	Conceito	Conceito	Conceito	Conceito	Conceito
Orzechowski et al. (2011)	Baixa	Alta	2	Não	Não
Seligman et al. (2010)	Baixa	Alta	3	Sim	Não
Gong e Chen (2010)	Baixa	Baixa	2	Não	Não
Goodall et al. (2008)	Alta	Baixa	2	Não	Não
Naidu et al. (2007)	Alta	Alta	1	Sim	Não
Lee e Jeong (2006)	Alta	Alta	1	Não	Não
Borkar (2004)	Baixa	Alta	3	Não	Não
Masseroli et al. (2004)	Alta	Alta	2	Não	Não
Draper et al. (2001)	Baixa	Alta	3	Não	Não
Thiran et al. (1998)	Alta	Alta	1	Sim	Não
Mannai e Bugrara (1993)	Alta	Alta	1	Não	Não
Marinos et al. (1991)	Conceito	Conceito	Conceito	Conceito	Conceito

Tabela 3.1: Características dos estudos

Após a execução do mapeamento sistemático, e análise da tabela comparativa, foi feita uma relação como forma de comparação com a proposta deste trabalho e seus contextos.

O objetivo desta etapa é buscar uma compreensão sobre os problemas que permeiam a ausência de integração entre sistemas, realizando uma análise das propostas de mecanismos integradores que, de alguma forma, entregaram respostas para seus respectivos desafios. Os trabalhos escolhidos foram separados e estudados de maneira a correlacionar os aspectos comuns dentre todos. Ao avaliar o quadro comparativo, derivado do mapeamento sistemático, foi possível identificar uma intersecção de propostas que têm uma baixa percepção de acoplamento, uma alta percepção de coesão e utiliza-se de múltiplas técnicas de integração.

Os destaques para este levantamento foram para Draper et al. (2001), Borkar e Vinayak (2004), Seligman et al. (2010) e Orzechowski e Dziech (2011). Seus estudos

foram bem construídos, propondo soluções completas principalmente no que diz respeito aos contextos em que foram trabalhados. A técnica da criação de um modelo de dados por *XML* em Draper et al. (2001) apresenta a vantagem de depender de menos infraestrutura para ser implementada, por ser um dos modelos de padrão de comunicação universal, além da ideia de quebrar uma consulta única em consultas nas bases heterogêneas apropriadas para cada segmento ter sido bem abordada. A abordagem de Borkar e Vinayak (2004) assemelha-se muito com a de Draper et al. (2001), portanto podendo promover o mesmo tipo de vantagens, pelo menos no que diz respeito às analogias de técnicas utilizadas, uma vez que os contextos onde foram inseridas são diferentes.

Dentre todos os estudos aqui relacionados, o de Seligman et al. (2010) é o que oferece mais recursos e mais se relaciona ao objetivo deste trabalho. Entretanto, pelo fato de realizar unificação semântica entre os dados de diferentes bases, não poderia ser aplicado como uma solução já pronta, uma vez que, para o tipo de integração que será abordado neste trabalho, não serão utilizados dados unificados, muito pelo contrário, os dados devem ser retornados diretamente de suas bases originais, seguindo seus próprios schemas.

O capítulo seguinte detalha a *InfraAPI* como solução de integração entre sistemas, expondo a *GiveMe Infra* e como ambas irão se relacionar.

4 Solução de Integração - *InfraAPI*

Considerando a importância e o estudo realizado sobre a automatização da troca de informações entre sistemas e tendo em vista a ausência desta automatização em alguns processos na *GiveMe Infra*, foi desenvolvida a *InfraAPI*, com o objetivo de convergir, em uma só estrutura, os métodos de integração entre fontes de dados e sistemas clientes, facilitando sua manutenção, aumentando sua coesão e diminuindo o seu acoplamento com relação à infraestrutura.

De maneira mais específica, promoverá o serviço de integração sobre os repositórios das ferramentas *Bugzilla e Redmine*, além de unificar algumas funcionalidades paralelas da *GiveMe Infra*. O motivo para esta escolha foi originado de uma solicitação do responsável e desenvolvedor da *GiveMe Infra*, além do fato de essas ferramentas pertencerem a um mesmo subgrupo de componentes de cadastro de defeitos de *software* que atendem ao *GiveMe Metrics framework*. A partir dessa integração a infraestrutura passará a contar com a disponibilização de um novo tipo de dado que antes não era utilizado, o de defeito de *software*.

Sua estrutura foi construída como um *plugin* do *Eclipse*, respeitando os padrões necessários para que um componente desse tipo seja compatível com o ambiente de execução, contendo arquivos de configuração, conhecido como manifesto que especificam seus atributos como, por exemplo, id, nome do fornecedor, versão, uma lista de *plugins* dos quais ele depende, e uma descrição para ser executado, além dos requisitos essenciais que um sistema codificado em Java precisa para ser implementado.

Pelo fato de ter sido idealizada como uma estrutura de apoio e automatização da da comunicação entre diferentes *softwares*, será introduzida a seguir a infraestrutura beneficiada e utilizada na prova de conceito.

4.1 *GiveMe Infra*

A *GiveMe Infra* é uma infraestrutura de suporte a tarefas de manutenção e evolução de *software*. Integrada ao *Eclipse* (Eclipse, 2017), é composta por várias ferramentas de apoio a equipes co-localizadas ou geograficamente distribuídas para execução dessas tarefas (Tavares, 2015). As ferramentas que a compõem são *plugins*, capazes de entregar diversas funcionalidades e artefatos, tais como cadastros de solicitação de mudança, versão do produto com a solicitação de mudança implementada, atividades de gerenciamento e manutenção de *software*.

Durante sua concepção, foram estabelecidos critérios para aceitação de seus componentes (i) Ou eles deveriam ser aptos a analisar dados da versão atual do código-fonte do *software* em desenvolvimento, extraindo variados tipos de informações sobre aquele projeto, tais como acoplamento entre classes e número de linhas de código (ii) Ou elas deveriam ser aptas a analisar dados históricos de um projeto, auxiliando o usuário na avaliação de informações de todas as fases do ciclo de manutenção do *software*. Além de pertencerem a um dos contextos, também deveriam ser capazes de fornecer recursos paralelos chamados de "recursos de colaboração" tais como capacidade de realizar troca de mensagens entre usuários, troca de mensagens sobre entidades analisadas durante algum processo e gerar *logs* de atividades no sistema.

Com a entrada utilizando dados do contexto atual, as ferramentas destinadas a analisar dados deste tipo têm o objetivo de auxiliar na compreensão de itens respectivos ao projeto de *software* e a inter-relação entre seus componentes (classes ou arquivos de código alterados).

Com a entrada utilizando dados do contexto histórico, as ferramentas destinadas a analisar dados deste tipo têm o objetivo de auxiliar na compreensão de itens respectivos às informações já cadastradas sobre os projetos, já descritas.

A seguir serão listados o conjunto das ferramentas de apoio, que compõem a *GiveMe Infra*:

- Analisador: uma ferramenta desenvolvida para dar base ao desenvolvimento do *Statistical Analysis Engine (SAE)* que é responsável por realizar análises estatísticas sobre rastreabilidade e impacto de alterações entre componentes, disponibilizando-as

para serem analisadas, neste contexto, pelo *plugin GiveMe Views*.

- *Sourceminer*: *plugin* que auxilia na compreensão das fases de manutenção e desenvolvimento de *software*, retornando métricas de código sob um ponto de vista gráfico (Carneiro, 2013). No contexto da *GiveMe Infra*, ele atua abrangendo contexto atual e histórico das fases.

- *ToolkitAIMV*: *toolkit* integrado ao *Sourceminer* fornecedor das diferentes abstrações visuais baseadas em compreensão de *software* que apoia a construção de ambientes interativos de múltiplas visões.

- *Colaborative Sourceminer*: *plugin* que possibilita a inserção de mensagens colaborativas nas visualizações geradas pelo *Sourceminer + AIMV*.

- *Collaboration Provider*: *plugin*, também desenvolvido no trabalho em questão que realiza o controle de acesso de *plugins* às informações do *Collaborative Sourceminer* via *web service*.

- *GiveMe Trace*: *plugin* desenvolvido por Lélis (2014) para o *Eclipse*, capaz de se conectar aos repositórios de controle de versão *GIT e SVN* e retornar informações sobre rastreabilidade nas modificações no código fonte, com granularidade fina (nível de método).

- *Mylyn Mantis*: no contexto da *GiveMe Infra*, este *plugin* é responsável pelo controle das solicitações de mudança dando origem às atividades de manutenção relacionadas a cada solicitação. As equipes distribuídas poderão executar as atividades e estas vão registrar as mudanças no código fonte ao final, possibilitando a fácil identificação das classes e métodos alterados pelo *GiveMe Trace*.

- *Subclipse*: no contexto do *GiveMe Infra*, este *plugin* é capaz de realizar o controle de versão, enviando informações ao repositório de código fonte após confirmação do desenvolvedor.

A figura 4.1 demonstra a forma de utilização da *GiveME Infra*

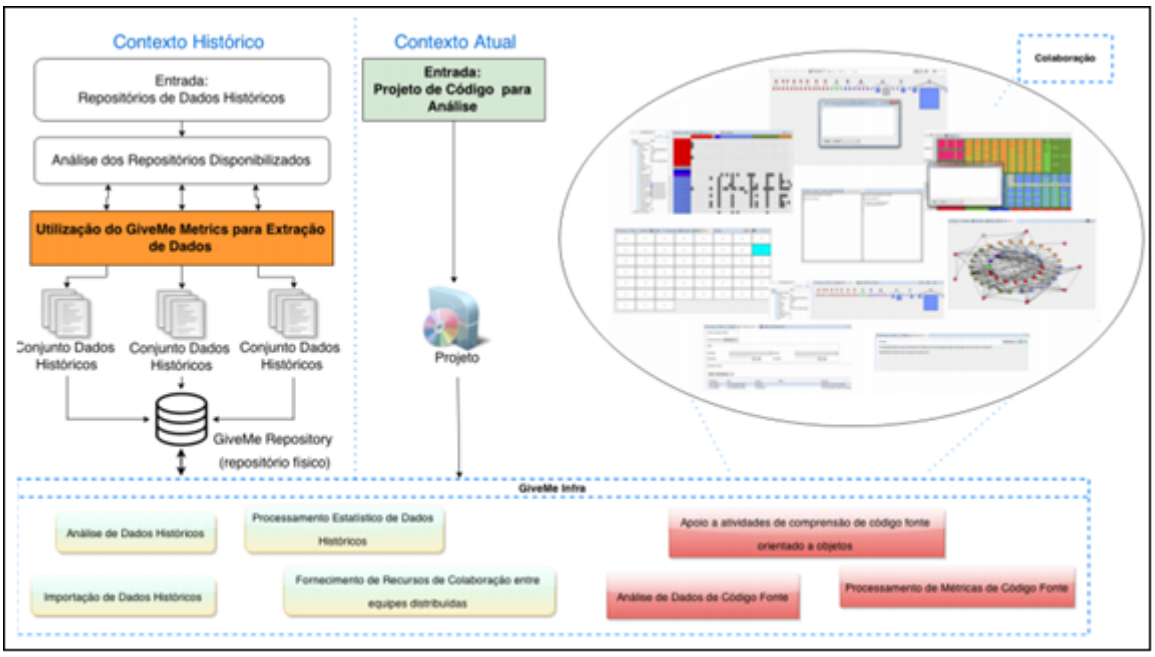


Figura 4.1: Uso dos recursos da *GiveMe Infra*

Além das ferramentas descritas acima, foram destacados especificamente o *GiveMe Metrics* e *GiveMe Views*, outros dois integrantes da infraestrutura que foram diretamente impactados pelo objeto de estudo deste trabalho, sendo interligados automaticamente.

4.1.1 *GiveMe Metrics*

Trata-se de um *framework* conceitual, montado a partir da união de ferramentas selecionadas com base em um mapeamento sistemático, realizado na fase inicial do projeto da infraestrutura. É capaz de apoiar a atividade de exportação de dados históricos provenientes de repositórios de código fonte, repositórios de defeitos ou repositórios de processos de desenvolvimento. Após exportadas, as informações são incluídas manualmente no *GiveMe Repository*, um outro componente responsável por gerenciar a criação de pastas de armazenamento automáticamente e (Tavares, 2015) e disponibilizá-las para serem importados pelas demais ferramentas que compõem a *GiveMe Infra* (Tavares et al, 2014). A partir disso, podem servir como base para a formação de variadas métricas e análises, dependendo da ferramenta componente que está trabalhando sobre aqueles dados no momento. A figura 4.2 esquematiza as opções e diretrizes de análise possíveis pela utilização do *framework*.

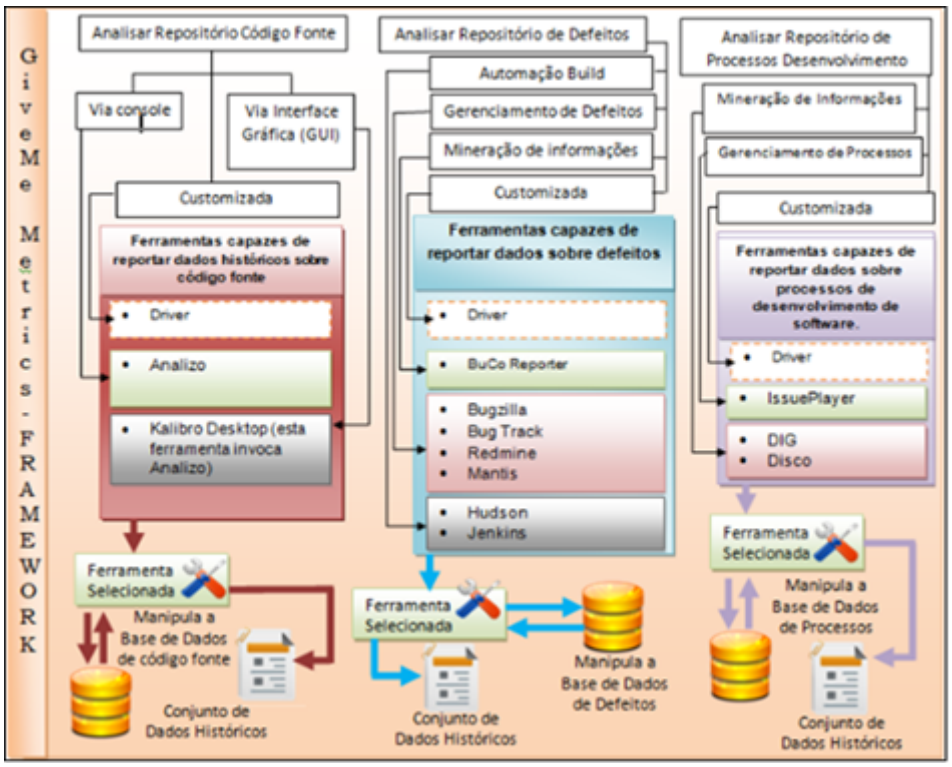


Figura 4.2: Visão geral do *GiveMe Metrics Framework*

Optando-se por analisar métricas de código fonte, o usuário pode ter como retorno as seguintes informações sobre código *Java* da versão escolhida: número de classes, número de métodos por classe, número de subsistemas, número de linhas de código, intervalo de tempo por versão de um projeto, profundidade de herança por classe, número de filhos por classe, acoplamento entre objetos, falta de coesão entre métodos e complexidade ciclomática.

Optando-se por analisar métricas de defeito de *software*, o usuário pode extrair informações sobre defeitos em aberto, defeitos resolvidos, autor dos defeitos, descrição do defeito, número de defeitos não resolvidos, número de defeitos resolvidos, tempo médio de correção de um defeito, idade média de um defeito não resolvido, número de defeitos detectados, depois do lançamento de uma *release*, relação entre defeitos resolvidos e não resolvidos, número de defeitos detectados depois de um período de tempo determinado e porcentagem de defeitos documentados. O tipo de dado retornado vai depender do tipo de base de dados de defeitos a ter seus dados exportados para análise.

Optando-se por analisar métricas de processos de desenvolvimento de *software*, o usuário pode ter como retorno as informações de quais são as tarefas concluídas ou

não concluídas, o tempo restante para a finalização das tarefas, esforço por membro da equipe no projeto, frequência absoluta, número máximo de repetições, duração total (não deixou explicito se considera duração de uma tarefa ou projeto), duração máxima, média de tempo dos artefatos no repositório, média de tempo dos defeitos cadastrados pertencentes a um projeto, média de tempo dos arquivos no repositório, média de tempo de construção das funcionalidades de um *software*.

4.1.2 *GiveMe Views*

Trata-se de um *plugin* para o *Eclipse* desenvolvido para ser integrante da GiveMe Infra capaz de atuar em análises sobre projetos de desenvolvimento de *software* em *Java* com o objetivo de dar suporte à fase de desenvolvimento de alguma alteração a ser executada (Tavares et al, 2015).

Para que que sua utilização seja efetiva, torna-se necessária a existência de uma massa de dados históricos disponíveis e armazenados no *GiveMe Repository*, um repositório de dados criado junto com a concepção desse *plugin*. Atualmente o *GiveMe Views* é capaz de gerar visualizações apenas sobre dados de código fonte e solicitação de mudanças extraídos por *drivers* implementados com finalidades específicas sobre o repositório.

Sua arquitetura compõe procedimentos que comportam processamento estatístico através da invocação da estrutura do *SAE*, persistência dos dados gerados no processamento estatístico, para embasar a geração das diferentes visões sobre as métricas ou aproveitamento por outras ferramentas.

Os passos de utilização da ferramenta são dados pela sequência: escolha do módulo e componente do projeto a serem alterados; importação dos dados históricos referentes a escolha, a partir do *GiveMe Repository*; filtragem dos dados históricos conforme regras de negócio previamente estabelecidas; execução das análises estatísticas, cálculo de métricas, persistência em memória dos dados estatísticos, métricas e outras informações da ferramenta; geração de visões baseadas nos dados persistido, ou integração com outras ferramentas; implementação de relatórios para exportação. A figura 4.3 elenca os passos implementados no *GiveMe Views*.

As evidências concretas de sua utilização se destacam nos módulos *Provedor* e

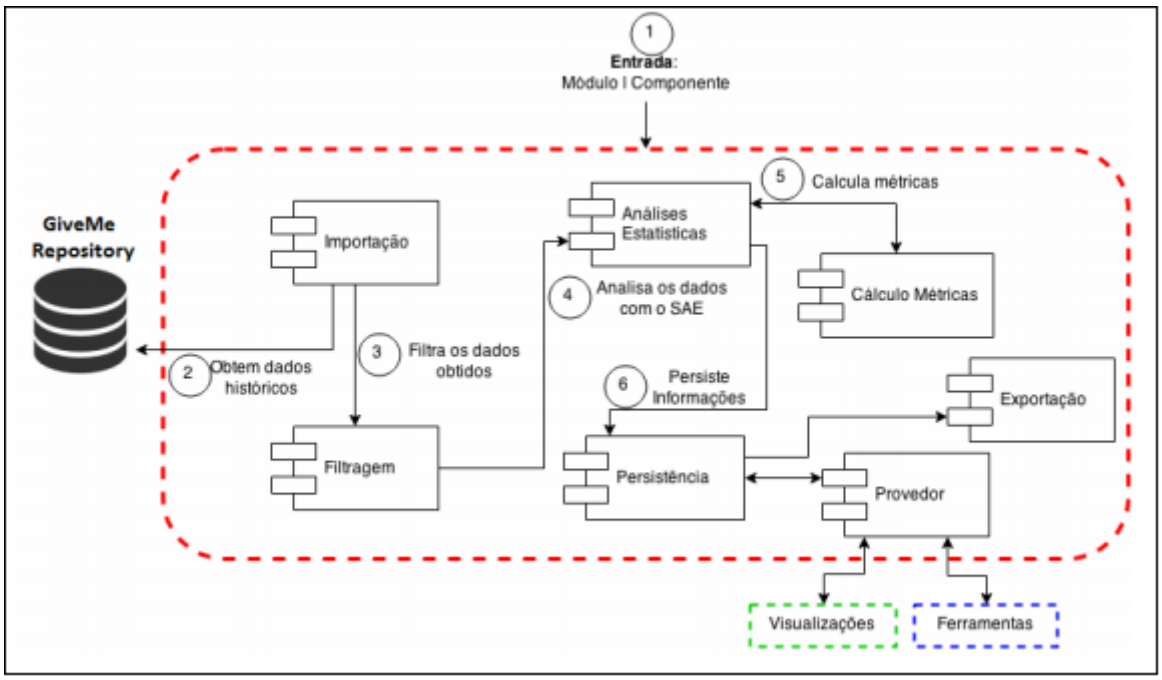


Figura 4.3: Visão geral do processo de utilização do *GiveMe Views*

Exportação, de sua arquitetura. Através da primeira é possível selecionar qual tipo de visualização de dados de código fonte ou históricos exibirá as relações de rastreabilidade ou impacto entre módulos ou componentes. A partir do segundo é possível exportar relatórios implementados. Em um momento posterior, O *GiveMe Views* foi integrado ao *GiveMe Trace*, aproveitando de sua capacidade de analisar os impactos e rastreabilidade de software com a granularidade mais fina. A Partir dessa parceria as visualizações contemplaram esta granularidade mais fina, exibindo métodos impactados, além dos módulos e componentes menos específicos.

Considerando toda a estrutura e da *GiveMe Infra* e, mais precisamente, a ausência de leituras e disponibilizações automáticas dos dados das bases históricas das ferramentas de defeitos de *software*, abrangidas pela *GiveMe Metrics*, será proposta a criação dos *drivers* para estas ferramentas, a fim melhorar a eficiência da troca de informações através da comunicação automatizada pela integração.

4.2 Arquitetura

A arquitetura da *InfraAPI* foi projetada para modularizar suas funcionalidades em camadas definidas com propósitos específicos, respeitando na essência o padrão de arquitetura

de software *Model-view-controller (MVC)*, das quais, 3 podem ser bem definidas a seguir:

- Camada de *Interface*: formada pelo pacote *infraapi.views*, responsável por receber o *input* de dados sobre troca de mensagens entre os desenvolvedores cadastrados na *GiveMe Infra*. Esta funcionalidade já existia na infraestrutura, porém foi deslocada para a *InfraAPI* com o intuito de iniciar um trabalho de unificação de serviços paralelos ao objetivo principal de sua utilização neste *plugin*, tentando torná-lo, em uma próxima versão, um elemento que retorna resultados para todo tipo de requisição por todos os componentes da *GiveMe Infra*.

- camada de *Controle*: formada pelos pacotes (i) *infraapi.filters*, responsável receber requisições por filtro para mensagens de usuários e (ii) *infraapi.provider*, responsável por controlar as requisições vindas da camada de interface, ou mesmo por disponibilizar métodos para acesso por outras aplicações que não dependem da existência de uma interface gráfica. Para o contexto do trabalho, esse último pacote representará a ligação mais importante com outros sistemas, através de sua classe *MasterDefectsData*, expondo as chamadas dos métodos para retorno das informações históricas. As funcionalidades presentes nesta classe são:

  - chamada para inclusão de novo registro de *bugs* no *Bugzilla* ou *Redmine*, relacionados a um determinado projeto.
  - chamada para recuperação de registros de *bugs* do *Bugzilla* ou *Redmine*, relacionados a um determinado projeto.
  - chamada para atualização de registros de *bugs* do *Bugzilla* ou *Redmine*, relacionados a um determinado projeto.
  - chamada para exclusão de registros de *bugs* no *Bugzilla* ou *Redmine*, relacionados a um determinado projeto.

- Camada de Persistência: formada pelos pacotes (i) *infraapi.persistence*, responsável por manter os métodos de conexão, consulta e disponibilização de fato dos dados entre repositórios escolhidos e sistemas consumidores dos dados, (ii) *infraapi.model*, responsável

Adicionalmente, existem os que foram denominados aqui de *Pacotes de Apoio*, representados por (i) *infraapi.valueObject*, sendo responsável por reunir os atributos de uma entidade em um único objeto permitindo, desta forma, que os métodos expostos recebam como argumento um tipo *value object* como parâmetro e não uma determinada quantidade de parâmetros, para recuperar objetos da base de dados, isto evita frequentes requisições na rede para acessar dados isolados, (ii) *infraapi*, responsável por ser a classe que possui os métodos de controle de ciclo de vida do *plugin* (execução, parada e instância), (iii) *infraapi.utility*, responsável por definir padrões de linguagem.

A figura 4.4 esquematiza a interdependência entre os pacotes que formam a *InfraAPI*.

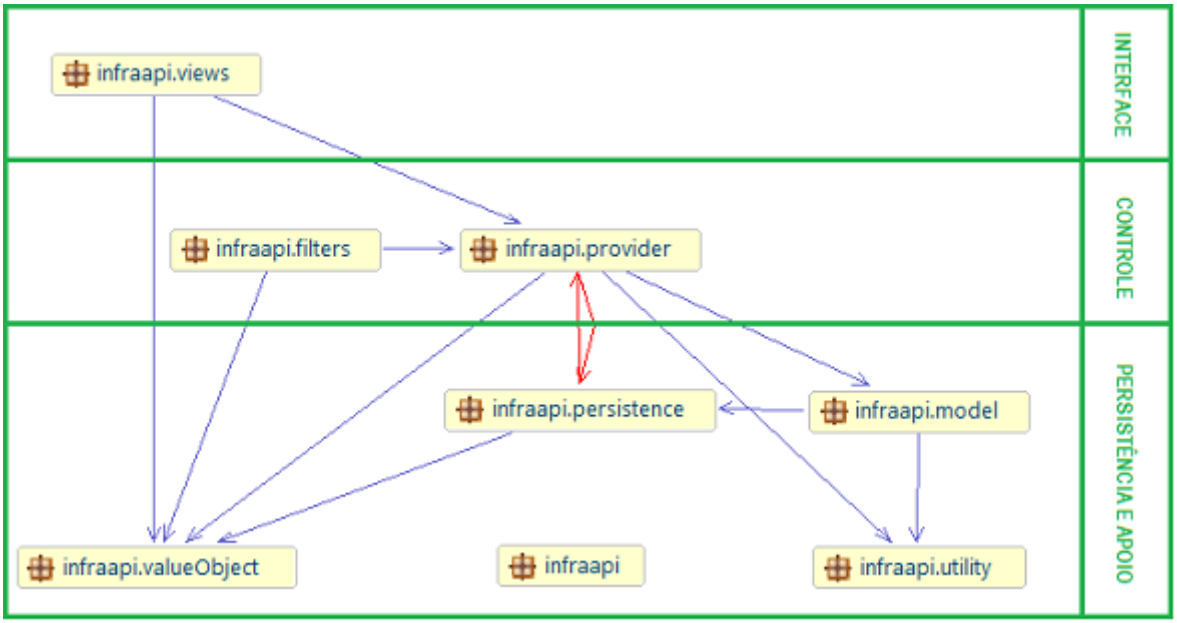


Figura 4.4: Visão geral do diagrama de dependências de pacotes da *InfraAPI*

A relação entre a *InfraAPI* e a *GiveMe infra* ocorre de maneira mais estreita através da *GiveMe Views*, uma vez que esse é o *plugin* integrante da infraestrutura responsável por ler dados históricos e armazenar na memória de execução através de seu módulo de importação, representados aqui pelos *bugs* relacionados a algum projeto *Java* a ser analisado.

A interação entre os dois *plugins* inicia no momento em que o usuário importa um projeto na janela de execução da *GiveMe Infra* (nova instância do *Workbench Eclipse*). Ao selecionar a opção *Visualize Historical Data Analysis (GiveMe Views)* no menu de contexto do projeto (módulo), são importados para a memória da infraestrutura, em tempo

de execução, os dados históricos referentes a todos os componentes daquele projeto. Essa memória da infraestrutura possui estruturas de dados responsáveis por manterem organizadas as informações importadas, para posteriormente serem retornadas na camada de interface gráfica, quando o usuário selecionar um dos componentes do projeto a ser analisado. A memória também já era carregada pelos *Drivers* implementados para os tipos de dados históricos da proposta original do *GiveMe Views*, que acessam o *GiveMe Repository* e fazem a leitura dos arquivos incluídos manualmente pelo usuário. A automatização realizada pela InfraAPI não utiliza o *GiveMe Repository*, sua execução carrega diretamente a memória com os dados recuperados das bases originais das ferramentas, filtrados pelo projeto escolhido, além de abranger um novo tipo de dados.

A figura 4.5 destaca a interação detalhada entre *GiveMe Views* e a *InfraAPI*.

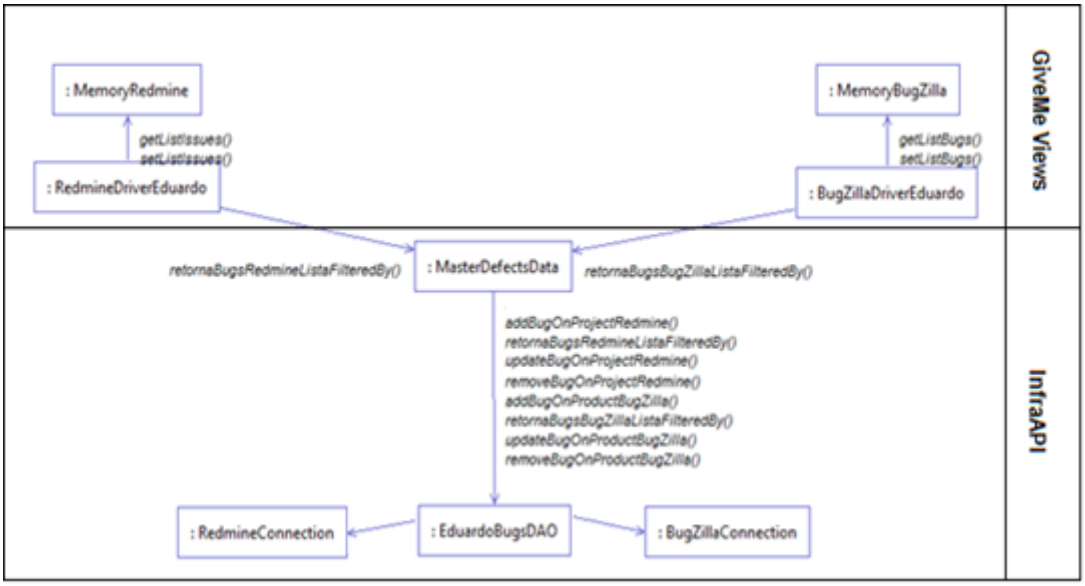


Figura 4.5: Cenário de interação entre o *GiveMe Views* e a *InfraAPI*

Com base no cenário descrito anteriormente, serão esquematizadas as formas com que os métodos implementados na *InfraAPI* se comportam.

4.2.1 Integração com *Bugzilla*

Bugzilla é um sistema web de cadastro, gerenciamento e rastreamento de defeitos e mudanças, que permite aos desenvolvedores ou grupos, um acompanhamento de *bugs* encontrados em produtos de software (Bugzilla, 2017). É possível realizar o download e a instalação dos seus componentes em um servidor web local com liberdade de escolha, desde

que ele seja capaz de executar *scripts Common Gateway Interface (CGI)*. Seu banco de dados também é instalado manualmente e pode ser escolhido entre *MySQL, PostgreSQL ou Oracle*. Aqui serão utilizados um servidor *web Apache* (Apache, 2017) e banco de dados *MySQL* (MySQL, 2017), por serem componentes abertos e de configuração facilitada.

A partir de sua instalação, seu banco de dados fica acessível aos usuários cadastrados, isto permite que outros sistemas possam se conectar na instância e realizar consultas diretas, caso esse acesso como usuário. Além desta liberdade de acesso aos dados, o *Bugzilla* ainda disponibiliza os dados via *Web services* para que sejam consumidos por sistemas externos que tenham acesso à ferramenta.

A integração entre a *InfraAPI* e o *Bugzilla* seguiu a proposta de *Wrapper* de banco de dados através do *driver JDBC* de conexão segura. Esta via de comunicação permite ao sistema cliente do banco de dados, no caso a *InfraAPI*, realizar todas as operações as quais tem permissão.

O seu processo de inclusão de dados através do mecanismo de integração segue os passos do diagrama 4.6, em que o *GiveMe Views* realiza a chamada de seu método de inclusão passando como parâmetro o *bug* que será incluído e o projeto ao qual o *bug* se referencia. Em seguida, a *InfraAPI* se conecta ao banco de dados relativo ao método chamado e insere o *bug* relacionado ao projeto, retornando "True.

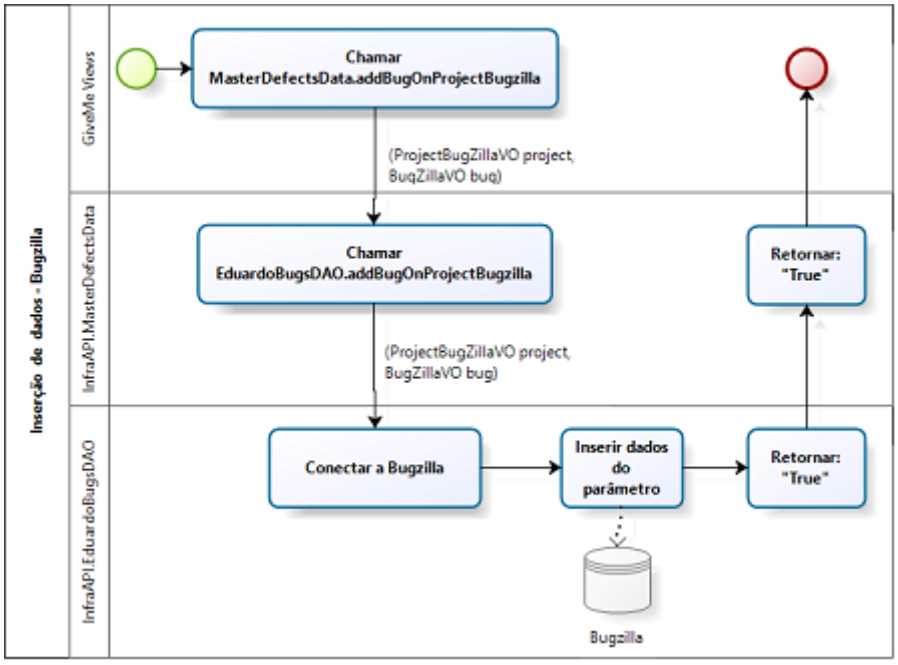


Figura 4.6: Inclusão automatizada de *bugs* na base do Bugzilla

O seu processo de recuperação de dados, através do mecanismo de integração, segue os passos do diagrama 4.7, em que o *GiveMe Views* realiza a chamada de seu método passando como parâmetro o endereço do banco de dados, o usuário, a senha e os filtros para recuperação dos *bugs* desejados. Em seguida, a *InfraAPI* se conecta ao banco de dados relativo ao método chamado, realiza a consulta conforme o parâmetro, retornando a lista para o *driver* que realizou a chamada do método.

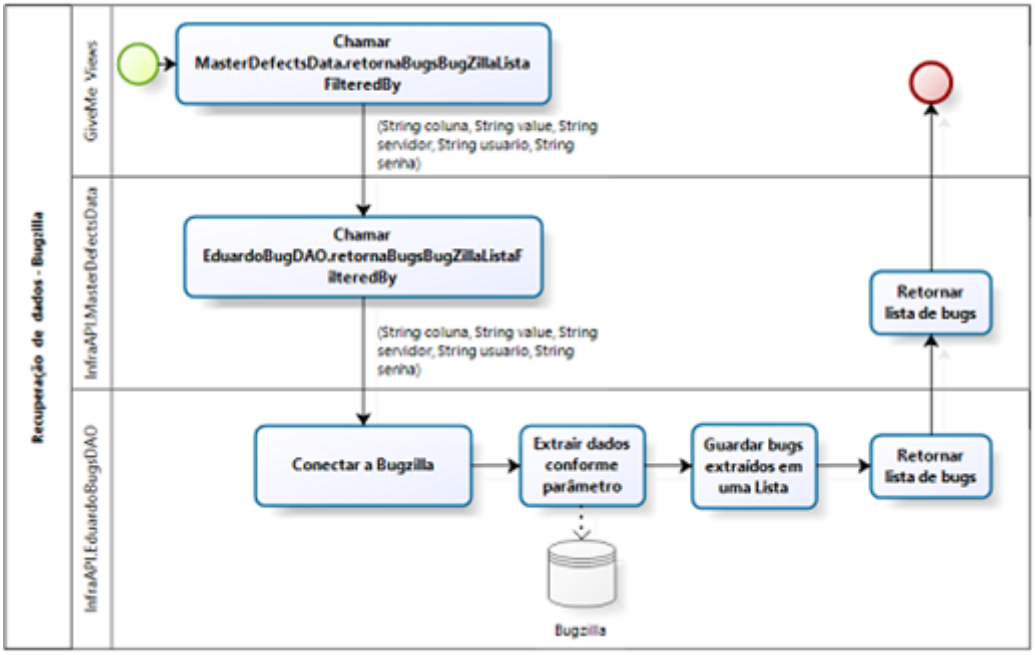


Figura 4.7: Recuperação automatizada de *bugs* da base do Bugzilla

O seu processo de atualização de dados, através do mecanismo de integração, segue os passos do diagrama 4.8, em que o *GiveMe Views* realiza a chamada de seu método passando como parâmetro o *bug* específico a ser atualizado, o projeto relacionado e o novo valor que o *bug* vai receber. Em seguida, a *InfraAPI* se conecta ao banco de dados relativo ao método chamado, realiza a atualização conforme o parâmetro passado, retornando um "True".

O seu processo de exclusão de dados, através do mecanismo de integração, segue os passos do diagrama 4.9, em que o *GiveMe Views* realiza a chamada de seu método de exclusão passando como parâmetro o *bug* que será removiido e o projeto ao qual o *bug* se referencia. Em seguida, a *InfraAPI* se conecta ao banco de dados relativo ao método chamado e remove o *bug* relacionado ao projeto, retornando "True".

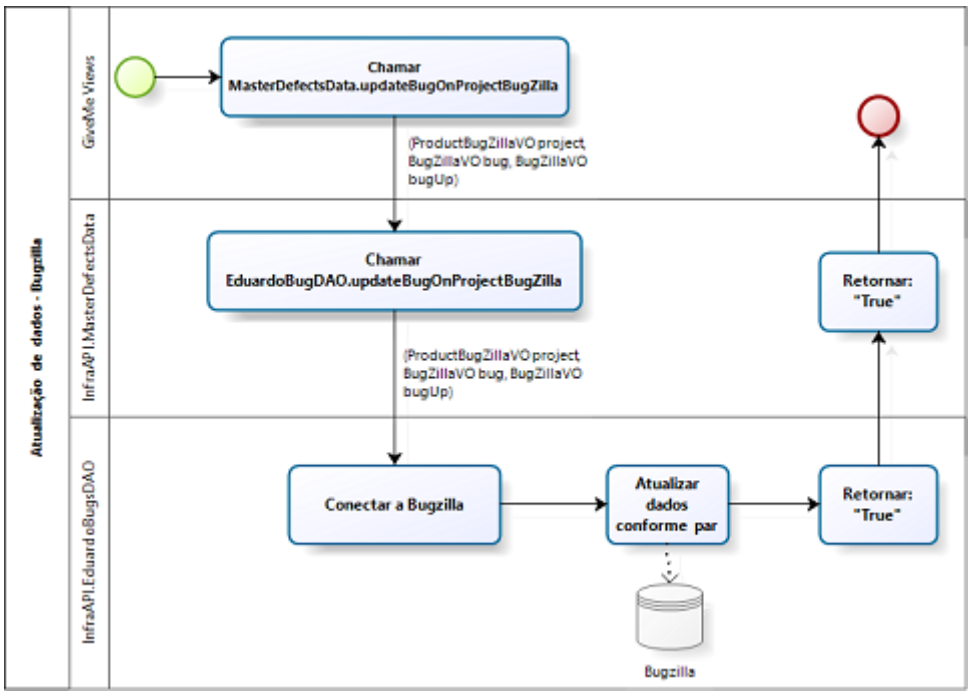


Figura 4.8: Atualização automatizada de *bugs* da base do Bugzilla

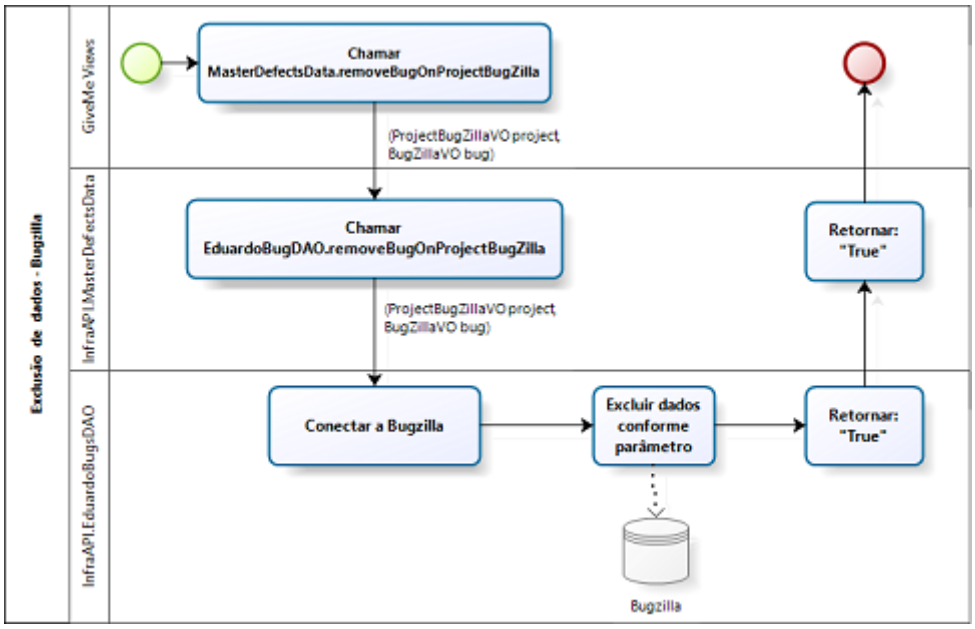


Figura 4.9: Exclusão automatizada de *bugs* da base do Bugzilla

4.2.2 Integração com *Redmine*

O *Redmine* é uma ferramenta web de gerenciamento de projetos (Redmine, 2017). Dentre suas funcionalidades existe a de gerenciamento de problemas, ou *bugs*. Os problemas podem estar associados a projetos e versões do produto de *software* que está cadastrado no sistema. Sua instalação é semelhante à do *Bugzilla*. É possível realizar o *download* e a instalação dos seus componentes no servidor web local *WEBrick ou no Apache*, também

capaz de executar scripts CGI. Seu banco de dados também é instalado manualmente e pode ser escolhido entre *MySQL, PostgreSQL ou Microsoft SQL Server*. Aqui serão utilizados um servidor web Apache (WEBrick, 2017) e banco de dados *MySQL* (MySQL, 2017), pelo mesmo motivo da abordagem com o *Bugzilla*.

Seu banco de dados também fica acessível aos usuários cadastrados, isto permite que outros sistemas possam se conectar na instância e realizar consultas diretas, caso tenham acesso como usuário. O *Redmine* também tem a opção de acesso a dados via *web services*.

A integração entre a *InfraAPI* e o *Redmine* seguiu a mesma proposta do *Bugzilla*, *Wrapper* de banco de dados através do *driver JDBC* de conexão segura.

O seu processo de inclusão de dados através do mecanismo de integração segue os passos do diagrama 4.10, em que o *GiveMe Views* realiza a chamada de seu método de inclusão passando como parâmetro o *bug* que será incluído e o projeto ao qual o *bug* se referencia. Em seguida, a *InfraAPI* se conecta ao banco de dados relativo ao método chamado e insere o *bug* relacionado ao projeto, retornando "True".

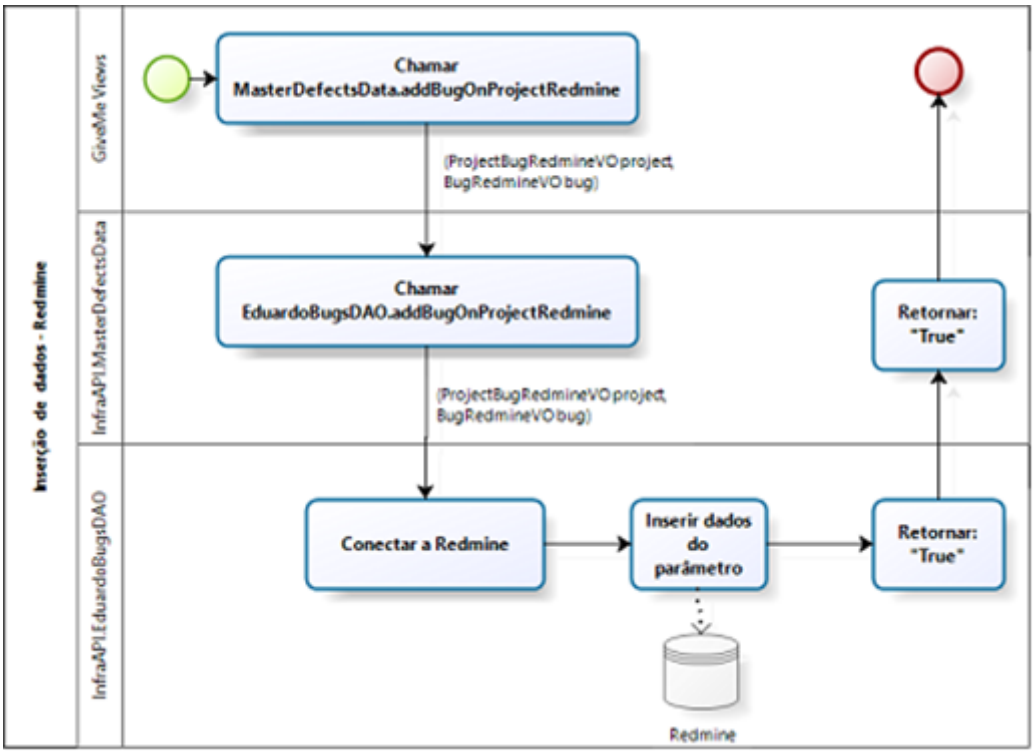


Figura 4.10: Inclusão automatizada de *bugs* na base do Redmine

O seu processo de recuperação de dados, através do mecanismo de integração, segue os passos do diagrama 4.11, em que o *GiveMe Views* realiza a chamada de seu

método passando como parâmetro o endereço do banco de dados, o usuário, a senha e os filtros para recuperação dos *bugs* desejados. Em seguida, a *InfraAPI* se conecta ao banco de dados relativo ao método chamado, realiza a consulta conforme o parâmetro, retornando a lista para o *driver* que realizou a chamada do método.

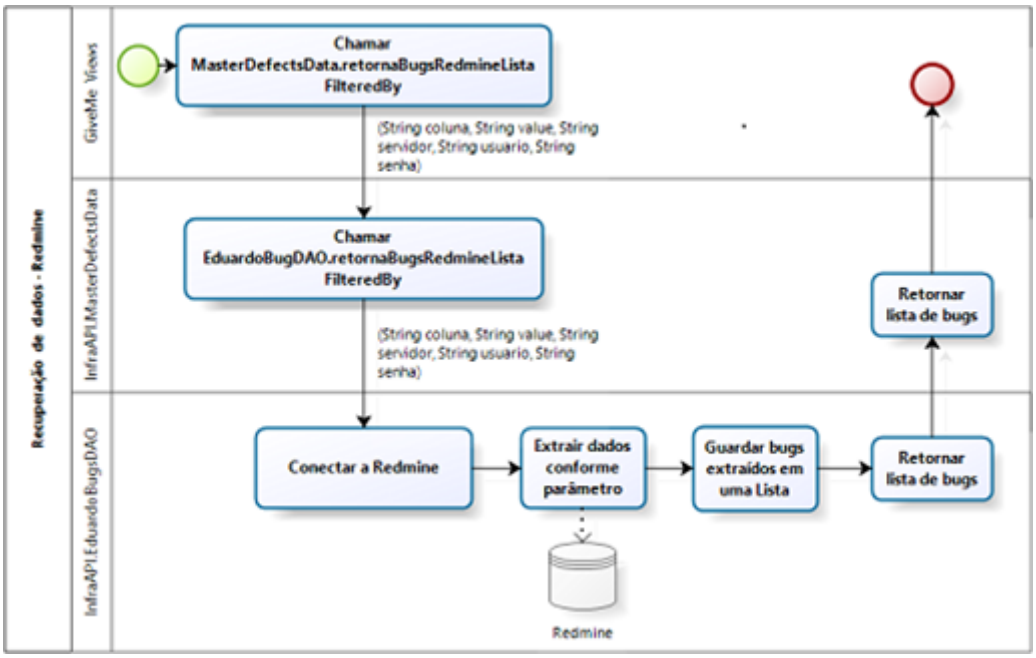


Figura 4.11: Recuperação automatizada de *bugs* da base do Redmine

O seu processo de atualização de dados, através do mecanismo de integração, segue os passos do diagrama 4.12, em que o *GiveMe Views* realiza a chamada de seu método passando como parâmetro o *bug* específico a ser atualizado, o projeto relacionado e o novo valor que o *bug* vai receber. Em seguida, a *InfraAPI* se conecta ao banco de dados relativo ao método chamado, realiza a atualização conforme o parâmetro passado, retornando um "True".

O seu processo de exclusão de dados, através do mecanismo de integração, segue os passos do diagrama 4.13, em que o *GiveMe Views* realiza a chamada de seu método de exclusão passando como parâmetro o *bug* que será removido e o projeto ao qual o *bug* se referencia. Em seguida, a *InfraAPI* se conecta ao banco de dados relativo ao método chamado e remove o *bug* relacionado ao projeto, retornando "True".

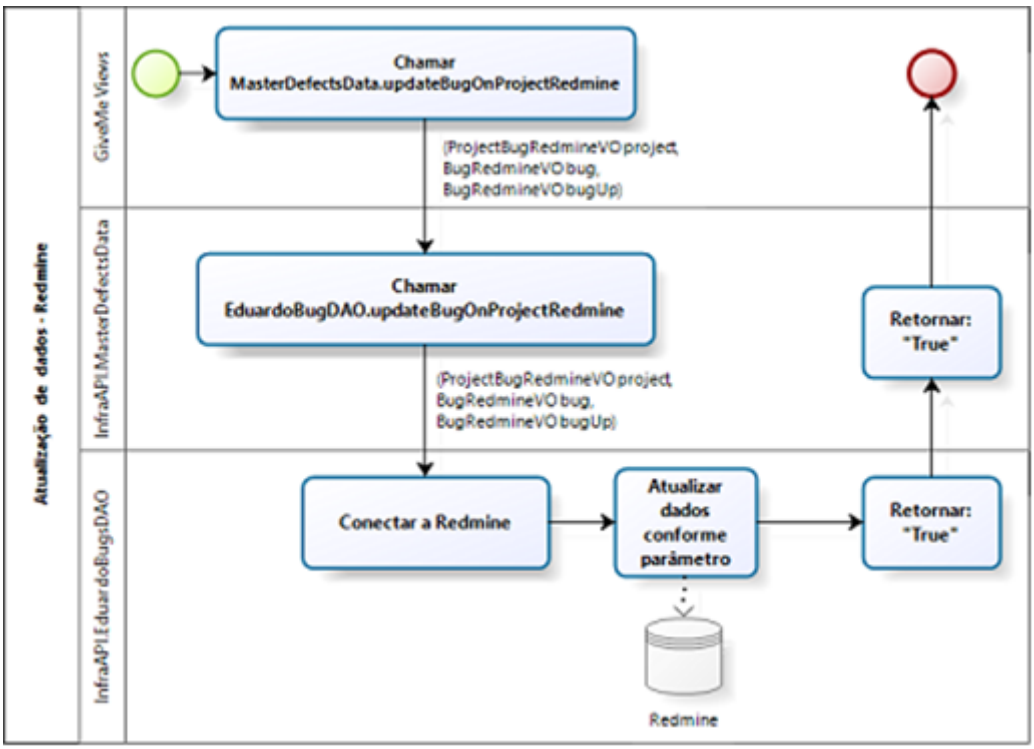


Figura 4.12: Atualização automatizada de *bugs* da base do Redmine

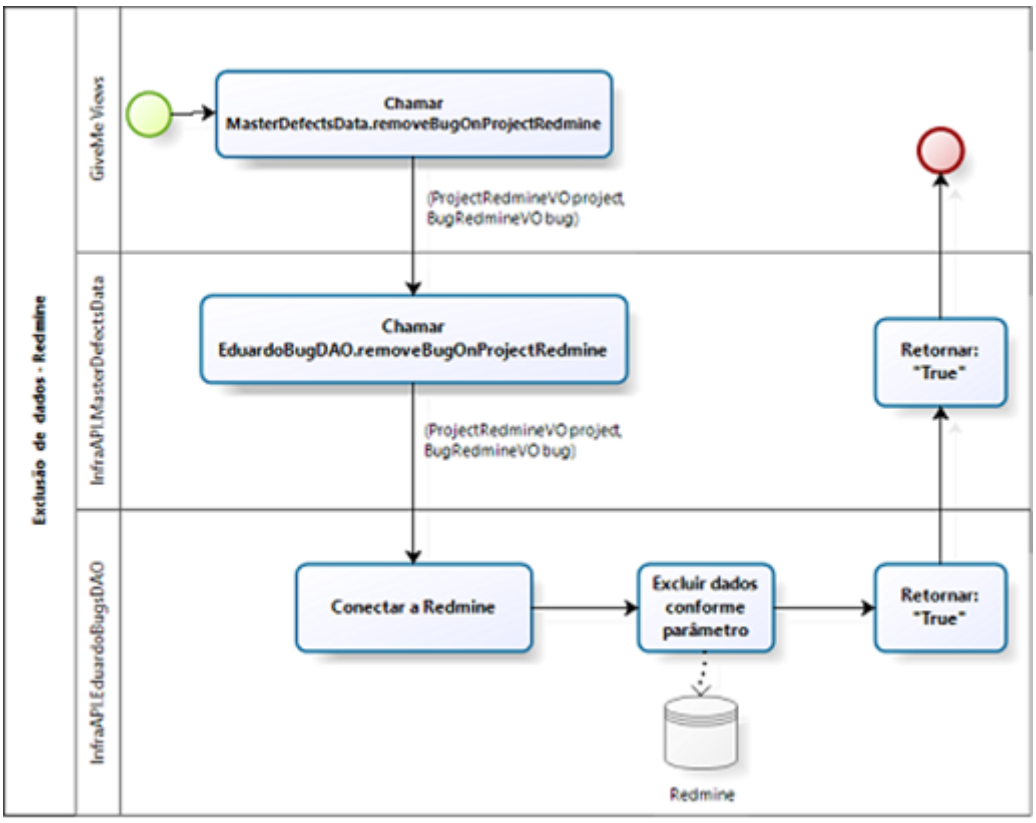


Figura 4.13: Exclusão automatizada de *bugs* da base do Redmine

Após a construção e instalação da InfraAPI, a visualização de todo o processo de troca de dados pôde ser representada como na figura 4.14.

A figura a 4.14 representa a estrutura da *GiveMe Infra* após a implementação da infra *API* com os *drivers*, esquematizando como passou as ser realizada a tramitação dos dados entre a infraestrutura e os repositórios de defeitos.

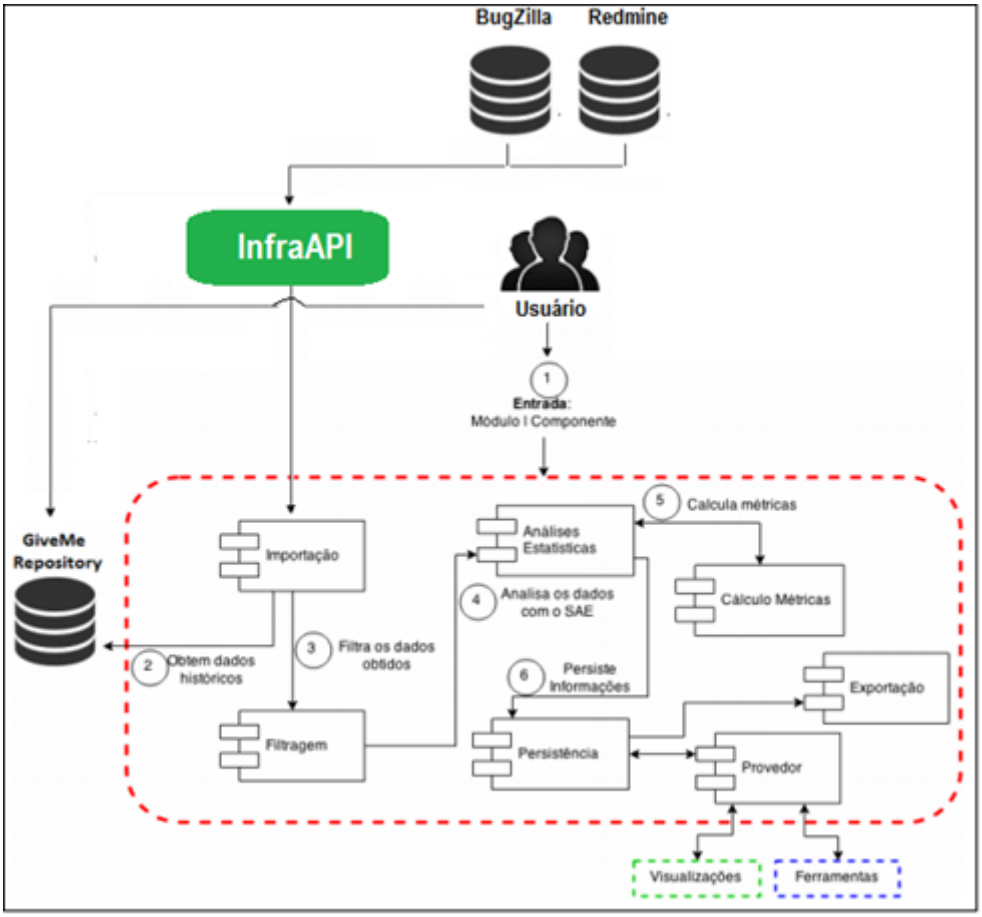


Figura 4.14: Cenário de utilização da *GiveMe Views* após a implementação dos *drivers*

A prévia descrição de sua arquitetura e utilização serviram para destacar a ideia de sua principal contribuição, que é promover um serviço de transferência de dados históricos em tempo de execução, a partir de repositórios de ferramentas de controle de defeitos com seus atributos definidos, sendo utilizados como fonte de informação para as análises finais realizadas pela *GiveMe Infra*. Sua execução é completamente transparente ao usuário, além do fato de ser capaz de se conectar ao banco de dados referenciado na chamada de seus métodos como argumento, bastando receber o endereço, usuário e senha para esta realização.

A divisão de sua arquitetura em camadas permite que novos componentes de integração sejam incluídos para novos *schemas* de bancos de dados de maneira mais fácil e intuitiva. Também permite que este *plugin* seja instalado em outros ambientes, visto que

ele funciona, principalmente, como uma *API*, ou seja, é desacoplada em relação ao local utilizado, dependendo somente da configuração interna da representação dos atributos da tabela do banco de dados a qual será realizada a consulta por seus métodos.

Sua construção também motivou a implementação dos *drivers* de leitura, propriamente ditos, na estrutura do *GiveMe Views* de forma complementar, permitindo que todo o processo de transferência de um ponto a outro fosse viabilizado, formando uma estrutura cooperativa.

Os novos *drivers* também representam um benefício, pois foram implementados sob uma nova forma, diferente dos já implementados no *GiveMe Views*, uma vez que a integração que eles auxiliam a promover ocorre de forma direta entre as bases de dados e a memória de execução, não passando pelo *GiveMe Repository* para ler arquirvos físicos que representem dados exportados manualmente pelo usuário.

O capítulo a seguir destaca a mecânica de execução da *InfraAPI* sob a ótica da prova de conceito, evidenciando os pontos de controle onde são carregados os dados, confirmando seu objetivo de construção.

5 Prova de Conceito

Este capítulo caracteriza a prova de conceito a qual a solução de integração foi aplicada, com a finalidade de validar a viabilidade e verificar os resultados obtidos com a automatização da troca de informações entre os repositórios de defeitos analisados e *GiveMe Infra*, através de casos de testes que caminham na execução do código da *InfraAPI* desde a chamada dos métodos de recuperação de dados até o retorno e carregamento da memória do *GiveMe Views* com os mesmos.

Os testes foram divididos em dois casos, cada um sendo executado sobre um cenário diferente de recuperação automática de dados. São eles:

- Caso 1: Chamada do método de recuperação de dados de bugs do *Bugzilla* na *InfraAPI*, pelo *GiveMe Views*.
- Caso 2: Chamada do método de recuperação de dados de bugs do *Redmine* na *InfraAPI*, pelo *GiveMe Views*.

Aplicou-se a prova de conceito utilizando dados de dois cenários reais.

O primeiro cenário escolhido foi a base de dados do *Bugzilla*, utilizada pelo *GNU Compiler Collection (GCC)*, um conjunto de compiladores que faz parte do projeto *GNU Is Not Unix (GNU)* para desenvolvimento de softwares e plataformas livres. Possui extensa relevância no contexto da computação pelo fato de ser utilizado como base para construção de variados sistemas operacionais, além do fato de ser portável em termos de aplicação em diferentes plataformas (GCC, 2017). Seu projeto é mantido e distribuído pela *Free Software Foundation (FSF)*, sob as diretrizes da *General Public License (GNU GPL)*, um tipo de licença que permite e encoraja aos usuários, de se utilizarem dos códigos fontes, estudarem e modificarem. Consequentemente, esses usuários podem realizar testes e reportar erros no produto, para isso disponibilizou-se o repositório Bugzilla, servindo como um ponto unificador de todas as descobertas de erros e registro de soluções para esses.

A importância do projeto GCC e a maturidade dos dados sobre defeito de software

já cadastrados no Bugzilla motivou a execução do caso 1 da Recuperação automática de dados, alertando que, para esse casoo caso, houve a necessidade de se realizar a cópia dos dados e inserção em um banco de dados local, também do *Bugzilla*. A justificativa para esta ação se dá pelo fato de não haver possibilidade de acesso direto para o banco de dados do *Bugzilla* utilizado pelo *GCC*.

O segundo cenário foi a base de dados do *Redmine*, utilizada pelo *Sistema Nacional de Processamento de alto Desempenho SINAPAD*. formado por uma rede de centros de computação, fundada pelo Ministério da Ciência e Tecnologia e Coordenada pelo Laboratório Nacional de Computação Científica (LNCC) (SINAPAD Redmine, 2017). Seu propósito é oferecer uma estrutura capaz de realizar alto processamento computacional sob demanda de diversas entidades, além de apoiar o constante desenvolvimento de tecnologias, transferir conhecimento e apoiar o desenvolvimento de pessoas especializadas em processamento de alto desempenho. Existe a probabilidade de que ocorram problemas nos produtos desenvolvidos nos centros, para isso foi disponibilizada essa base do Redmine para que sejam reportados ou tenha soluções de problemas registradas.

Toda estrutura montada para dar suporte às praticas no SINAPAD, também evidenciou a maturidade com que seus dados são manipulados, e foram com esses dados que se realizaram o Caso 2 da recuperação automática de dados, alertando também que, para esse caso, houve a necessidade de se realizar a cópia dos dados e inserção em um banco de dados local, do *Redmine*. A justificativa para esta ação se dá pelo fato de, assim como na base de dados do *Bugzilla*, não haver possibilidade de acesso direto para o banco de dados do *Redmine* utilizado pelo *SINAPAD*.

A exibição dos resultados sobre os dados carregados pode ser feita pela ferramenta *GiveMe Views*. A automatização da integração, exposta nos casos de testes, foi direcionada para analisar a efetividade do transporte dos dados de um ponto a outro do processo de execução. O detalhamento dos casos de testes segue nas próximas seções.

Conforme já mencionado, foram considerados nos casos de testes apenas métodos de recuperação de dados por parte do GiveMe Views, pelo fato de serem os únicos que retornam conjuntos de dados, e não apenas uma confirmação booleana.

Os dados para análise foram previamente inseridos nos bancos de dados do *Bug-*

zilla e Redmine, copiados fielmente das bases dos cenários reais do *GCC e SINAPAD*.

5.1 Caso 1

O desenvolvimento desse caso de teste ocorreu com o intuito de detalhar a forma automática de disponibilização de dados do *Bugzilla*, através da realização da chamada do método de recuperação na *InfraAPI* pelo *GiveMe Views*. Espera-se que os dados sejam mantidos íntegros durante sua transferência, e o número de etapas para que sejam recuperados na origem e inseridos no destino diminua.

As figuras 5.1 e 5.2 evidenciam, respectivamente, a fidelidade com que dez registros, utilizados como amostra neste caso de teste, foram replicados da página do *BugZilla GCC* para o banco de dados local.

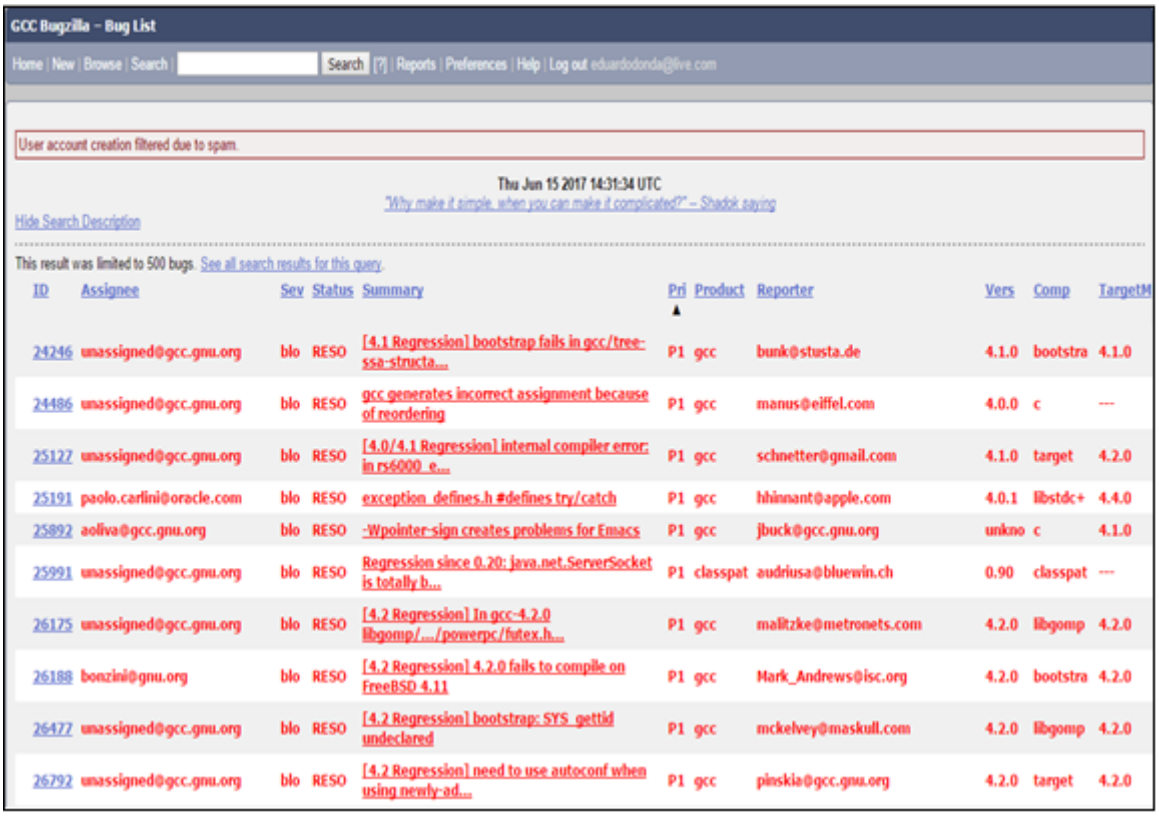
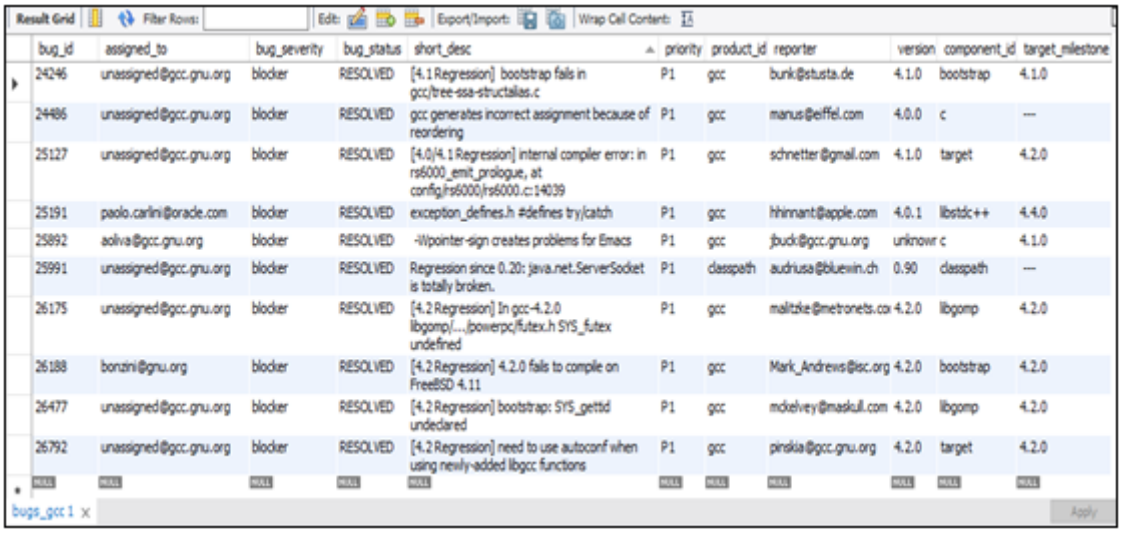


Figura 5.1: *Bugs* cadastrados no *Bugzilla GCC*

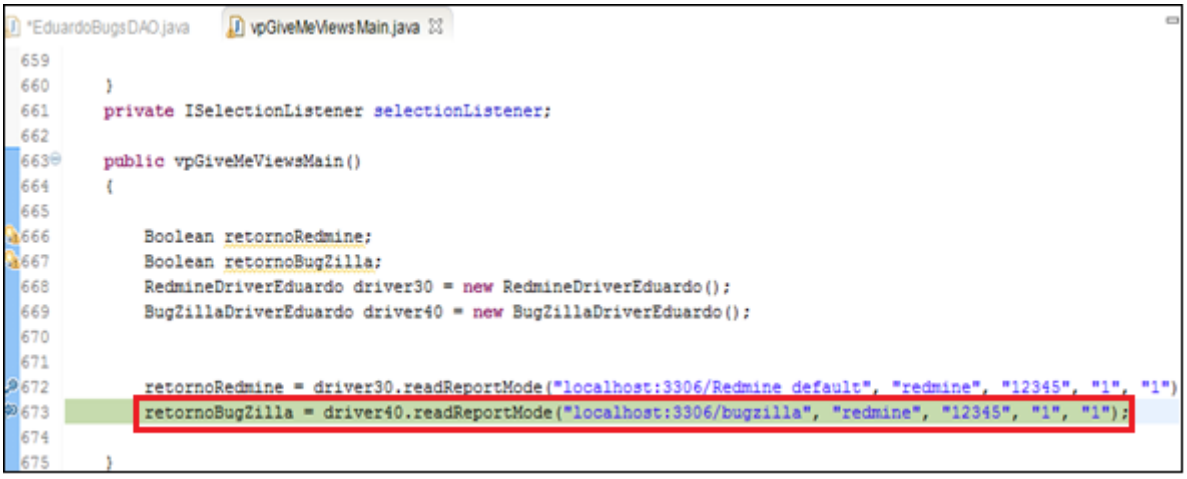


bug_id	assigned_to	bug_severity	bug_status	short_desc	priority	product_id	reporter	version	component_id	target_milestone
24246	unassigned@gcc.gnu.org	blocker	RESOLVED	[4.1 Regression] bootstrap fails in gcc/tree-ssa-structalias.c	P1	gcc	bunk@stusta.de	4.1.0	bootstrap	4.1.0
24486	unassigned@gcc.gnu.org	blocker	RESOLVED	gcc generates incorrect assignment because of reordering	P1	gcc	manus@effel.com	4.0.0	c	---
25127	unassigned@gcc.gnu.org	blocker	RESOLVED	[4.0/4.1 Regression] internal compiler error: in rs6000_emit_prologue, at config/rs6000/rs6000.c:14039	P1	gcc	schnetter@gmail.com	4.1.0	target	4.2.0
25191	paolo.carlini@oracle.com	blocker	RESOLVED	exception_defines.h #defines try/catch	P1	gcc	hhinnant@apple.com	4.0.1	libstdc++	4.4.0
25892	aoliva@gcc.gnu.org	blocker	RESOLVED	-Wpointer-sign creates problems for Emacs	P1	gcc	jbuck@gcc.gnu.org	unknowr	c	4.1.0
25991	unassigned@gcc.gnu.org	blocker	RESOLVED	Regression since 0.20: java.net.ServerSocket is totally broken.	P1	classpath	audriusa@bluewin.ch	0.90	classpath	---
26175	unassigned@gcc.gnu.org	blocker	RESOLVED	[4.2 Regression] In gcc-4.2.0 libgomp/.../powerpc/futex.h SYS_futex undefined	P1	gcc	malitzke@metronets.co	4.2.0	libgomp	4.2.0
26188	bonzini@gnu.org	blocker	RESOLVED	[4.2 Regression] 4.2.0 fails to compile on FreeBSD 4.11	P1	gcc	Mark_Andrews@isc.org	4.2.0	bootstrap	4.2.0
26477	unassigned@gcc.gnu.org	blocker	RESOLVED	[4.2 Regression] bootstrap: SYS_gettid undeclared	P1	gcc	mckelvey@maskull.com	4.2.0	libgomp	4.2.0
26792	unassigned@gcc.gnu.org	blocker	RESOLVED	[4.2 Regression] need to use autoconf when using newly-added libgcc functions	P1	gcc	pinskia@gcc.gnu.org	4.2.0	target	4.2.0

Figura 5.2: *Bugs* cadastrados no Banco de dados do *Bugzilla GCC*

Como primeiro passo da execução, foram criadas no *GiveMe Views* as classes *BugZillaDriverEduardo* e *MemoryBugZilla*, onde foram implementados os métodos que realizam as chamadas de recuperação de dados na *InfraAPI* e carregam na memória de execução, que é disponibilizada para a *GiveMe Infra*, com dados relativos ao *Bugzilla*.

Seguindo, executou-se a *GiveMe Infra* em modo de *debug* para interceptar o momento em que os dados foram recuperados. A chamada que desencadeia esse processo de recuperação foi incluída na classe principal do *GiveMe Views*, conforme é evidenciado pela figura 5.3.



```
659
660     }
661     private ISelectionListener selectionListener;
662
663     public vpGiveMeViewsMain()
664     {
665
666         Boolean retornoRedmine;
667         Boolean retornoBugZilla;
668         RedmineDriverEduardo driver30 = new RedmineDriverEduardo();
669         BugZillaDriverEduardo driver40 = new BugZillaDriverEduardo();
670
671
672         retornoRedmine = driver30.readReportMode("localhost:3306/Redmine default", "redmine", "12345", "1", "1")
673         retornoBugZilla = driver40.readReportMode("localhost:3306/bugzilla", "redmine", "12345", "1", "1");
674
675     }
```

Figura 5.3: Chamada inicial da classe principal do *GiveMe Views* para recuperação de dados automática no *Bugzilla GCC*

A chamada aponta para a classe que implementa o *Driver* criado para realizar a leitura da base do *Bugzilla* na execução desse caso de teste, passando o endereço do banco

de dados, usuário, senha e campos a serem filtrados, caso necessário. As figuras 5.4 e 5.5 destacam a chamada criada no *driver* para a *InfraAPI* iniciar sua função de extração.

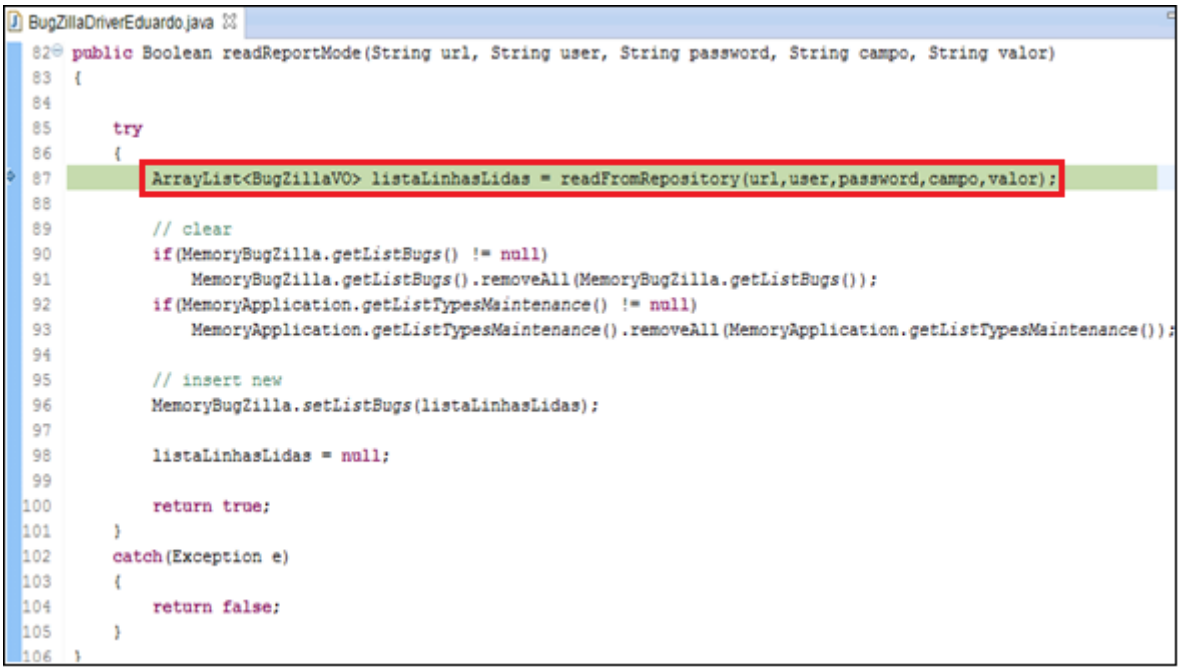


Figura 5.4: Chamada do método que vai consultar a *InfraAPI*

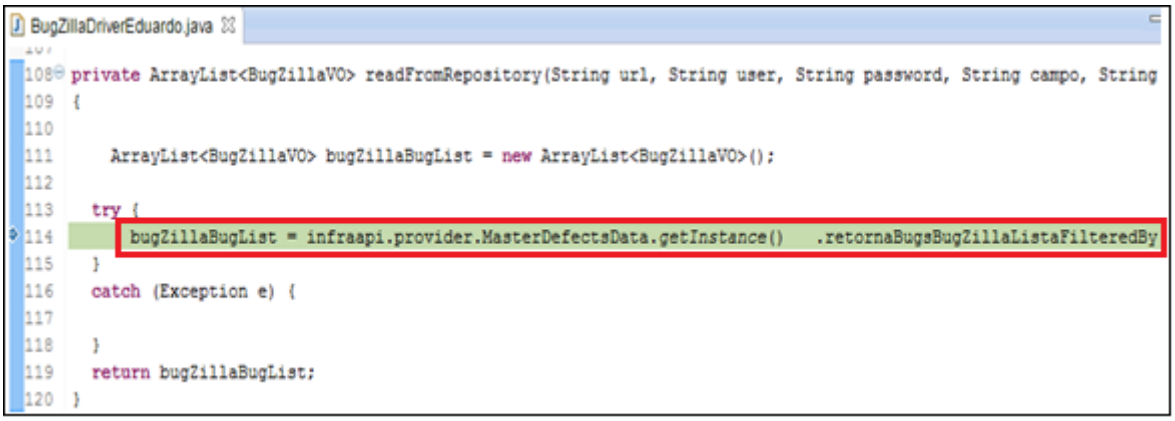


Figura 5.5: Método que consulta a *InfraAPI*

O próximo passo foi representado pela criação da lista de defeitos do *Bugzilla* com os registros presentes em seu banco de dados, evidenciados pela figura 5.6.

A seguir, o *driver* recebe a lista, anula a memória de execução e a reinicializa, passando a mesma lista para ser ali carregada, conforme esquematizado na figura 5.7

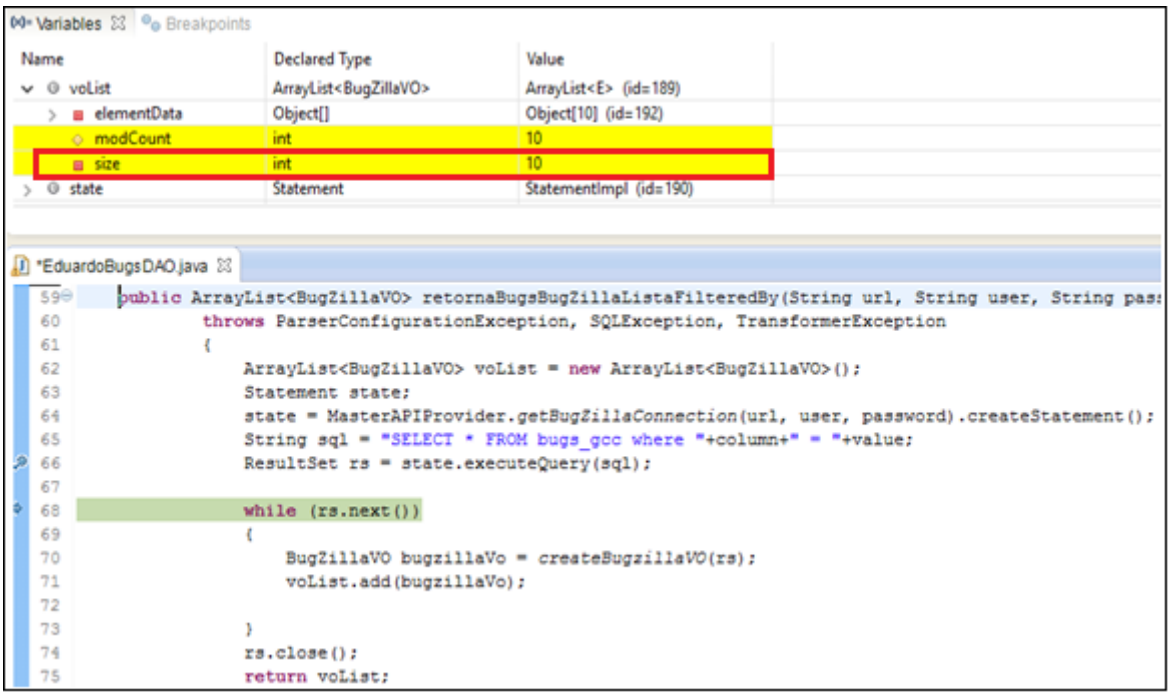


Figura 5.6: Retorno da consulta ao banco de dados do *Bugzilla GCC*

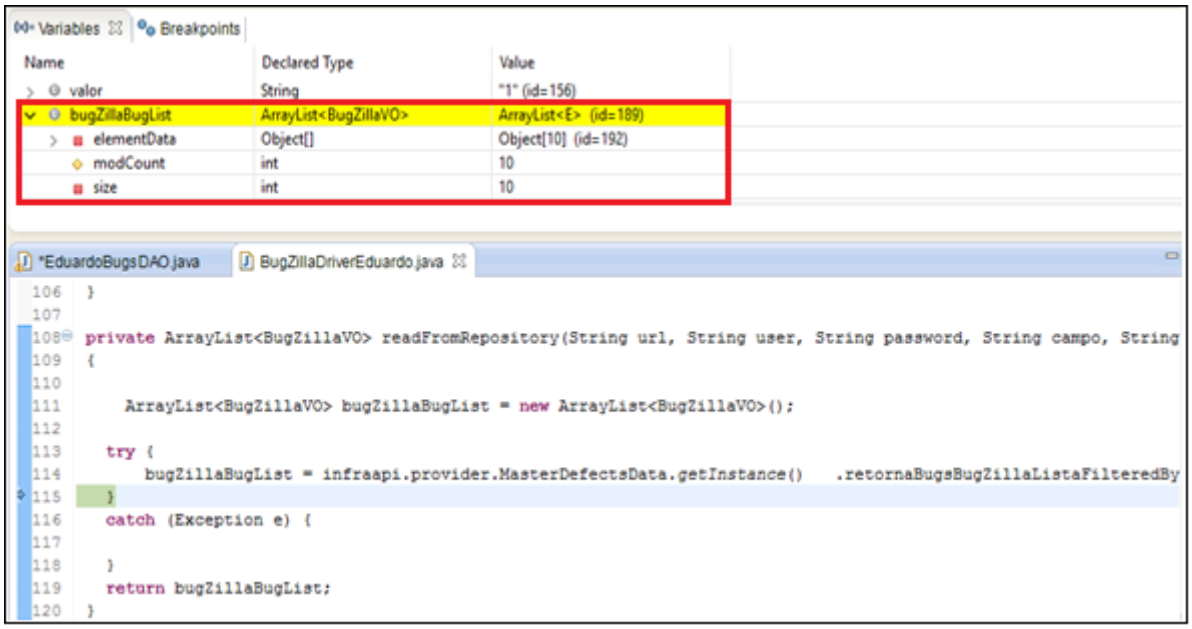


Figura 5.7: Retorno da lista de dados do *Bugzilla GCC* dentro do *driver*

Finalizando o processo, a figura 5.8 destaca a memória de execução do *Bugzilla* no *GiveMe Views* carregada com os mesmos registros presentes no banco de dados, disponíveis para serem analisados pelas ferramentas da *GiveMe Infra*.

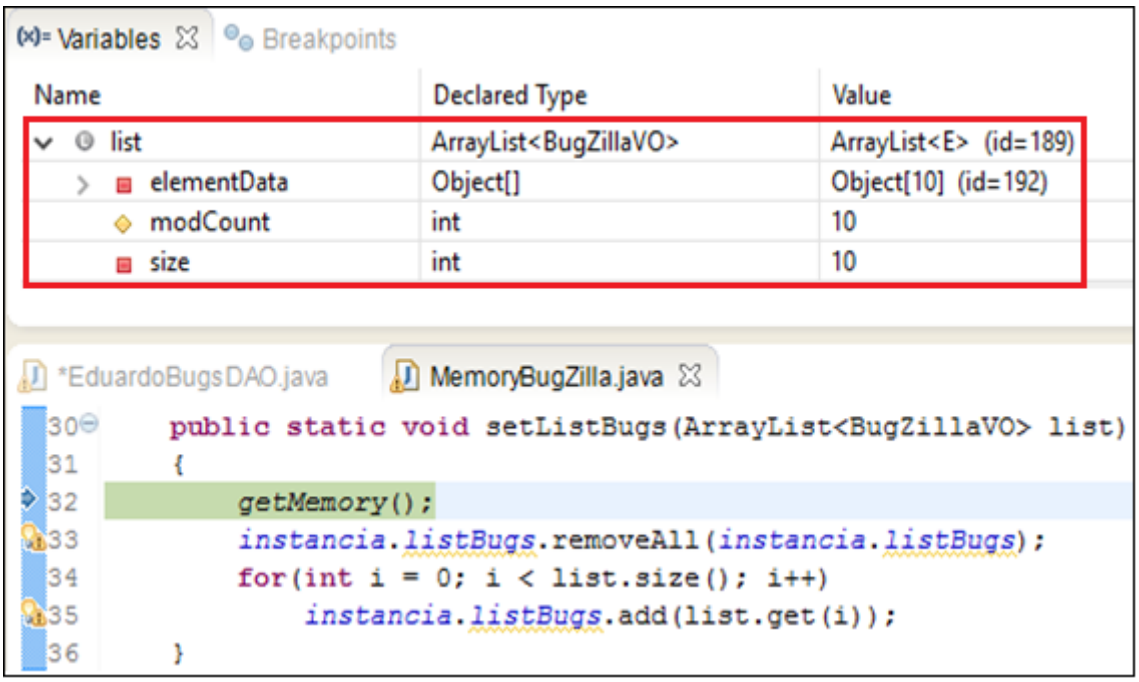


Figura 5.8: Memória de execução da *GiveMe Views* carregada com registros

A execução desse caso de teste evidenciou que não há o risco de ocorrência de alteração ou duplicação de dados, entre as etapas de chamada na *InfraAPI*, retorno de informações e carga na memória de execução. Os dados embasarão análises e métricas fiéis ao que, de fato, foi cadastrado na ferramenta de gerenciamento de defeitos do projeto. Ficou claro também que o processo de automatização ocorre em uma única etapa para o usuário final, o que diminui sua complexidade e encurta o processo de análise.

5.2 Caso 2

De forma análoga ao caso anterior ao desenvolvimento desse também ocorreu com o intuito de detalhar a forma automática de disponibilização de dados do *Redmine*, através da realização da chamada do método de recuperação na *InfraAPI* pelo *GiveMe Views*. Espera-se que os dados sejam mantidos íntegros durante sua transferência, e o número de etapas para que sejam recuperados na origem e inseridos no destino diminua.

As figuras 5.9 e 5.10 evidenciam, respectivamente, a fidelidade com que nove registros, utilizados como amostra neste caso de teste, foram replicados do *Redmine Sinapad* para o banco de dados local.

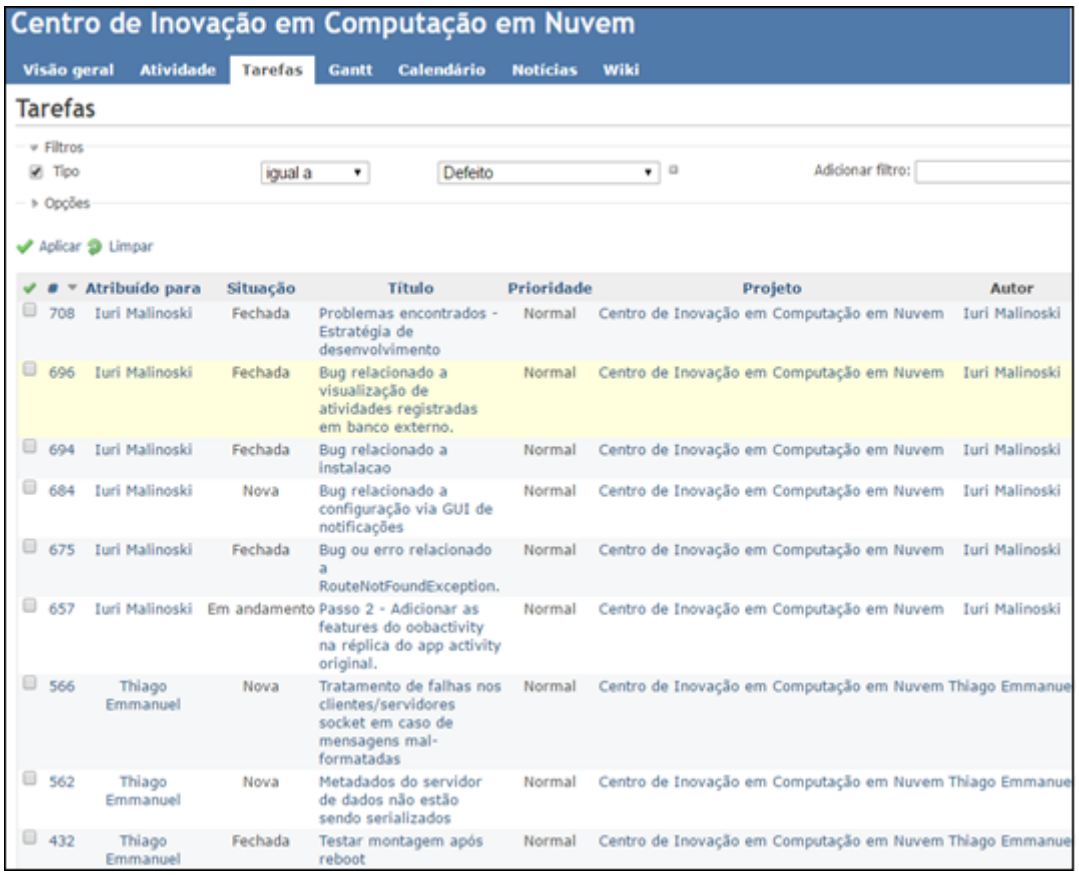


Figura 5.9: *Bugs* cadastrados no *Redmine SINAPAD*

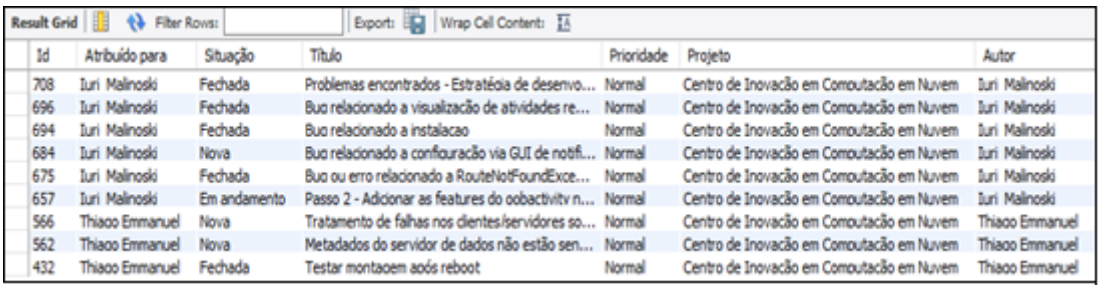


Figura 5.10: *Bugs* cadastrados no Banco de dados do *Redmine SINAPAD*

Como primeiro passo da execução, foram criadas no *GiveMe Views* as classes *RedmineDriverEduardo* e *MemoryRedmine*, onde foram implementados os métodos que realizam as chamadas de recuperação de dados na *InfraAPI* e carregam na memória de execução, que é disponibilizada para a *GiveMe Infra*, com dados relativos ao *Redmine*.

Seguindo, executou-se a *GiveMe Infra* em modo de *debug* para interceptar o momento em que os dados foram recuperados. A chamada que desencadeia esse processo de recuperação foi incluída na classe principal do *GiveMe Views*, conforme é evidenciado pela figura 5.11.

```
vpGiveMeViewsMain.java
658         }
659
660     }
661     private ISelectionListener selectionListener;
662
663     public vpGiveMeViewsMain()
664     {
665
666         Boolean retornoRedmine;
667         Boolean retornoBugZilla;
668         RedmineDriverEduardo driver30 = new RedmineDriverEduardo();
669         BugZillaDriverEduardo driver40 = new BugZillaDriverEduardo();
670
671
672         retornoRedmine = driver30.readReportMode("localhost:3306/Redmine_default", "redmine", "12345", "1", "1");
673         retornoBugZilla = driver40.readReportMode("localhost:3306/bugzilla", "redmine", "12345", "1", "1");
674
675     }
```

Figura 5.11: Chamada inicial da classe principal do *GiveMe Views* para recuperação de dados automática no *Redmine SINAPAD*

A chamada aponta para a classe que implementa o *Driver* criado para realizar a leitura da base do *Redmine* na execução desse caso de teste, passando o endereço do banco de dados, usuário, senha e campos a serem filtrados, caso necessário. As figuras 5.12 e 5.13 destacam a chamada criada no *driver* para a *InfraAPI* iniciar sua função de extração.

```
RedmineDriverEduardo.java
80
81      public Boolean readReportMode(String url, String user, String password, String campo, String valor)
82      {
83
84          try
85          {
86              ArrayList<BugRedmineVO> listaLinhasLidas = readFromRepository(url,user,password,campo,valor);
87
88              // clear
89              if(MemoryRedmine.getListIssues() != null)
90                  MemoryRedmine.getListIssues().removeAll(MemoryRedmine.getListIssues());
91              if(MemoryApplication.getListTypesMaintenance() != null)
92                  MemoryApplication.getListTypesMaintenance().removeAll(MemoryApplication.getListTypesMaintenance());
93
94              // insert new
95              MemoryRedmine.setListIssues(listaLinhasLidas);
96
97              listaLinhasLidas = null;
98
99              return true;
100         }
101         catch(Exception e)
102         {
103             return false;
104         }
```

Figura 5.12: Chamada do método que vai consultar a *InfraAPI*

O próximo passo foi representado pela criação da lista de defeitos do *Redmine* com os registros presentes em seu banco de dados, evidenciados pela figura 5.14.

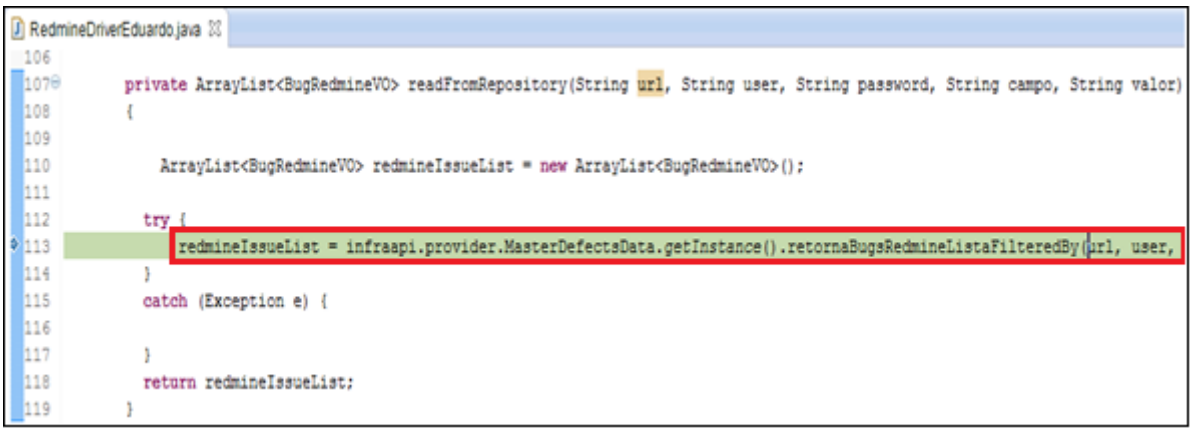


Figura 5.13: Método que consulta a *InfraAPI*

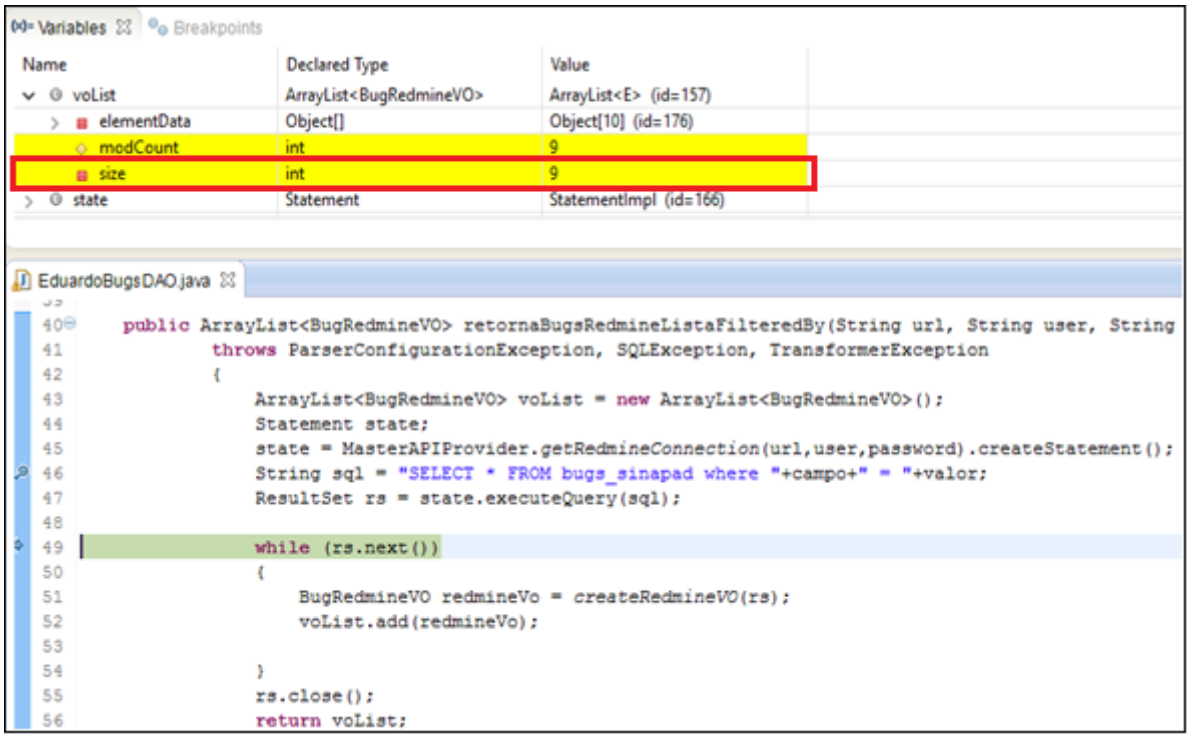


Figura 5.14: Retorno da consulta ao banco de dados do *Redmine SINAPAD*

A seguir, o *driver* recebe a lista, anula a memória de execução e a reinicializa, passando a mesma lista para ser ali carregada, conforme esquematizado na figura 5.15

Finalizando o processo, a figura 5.16 destaca a memória de execução do *Redmine* no *GiveMe Views* carregada com os mesmos registros presentes no banco de dados, disponíveis para serem analisados pelas ferramentas da *GiveMe Infra*.

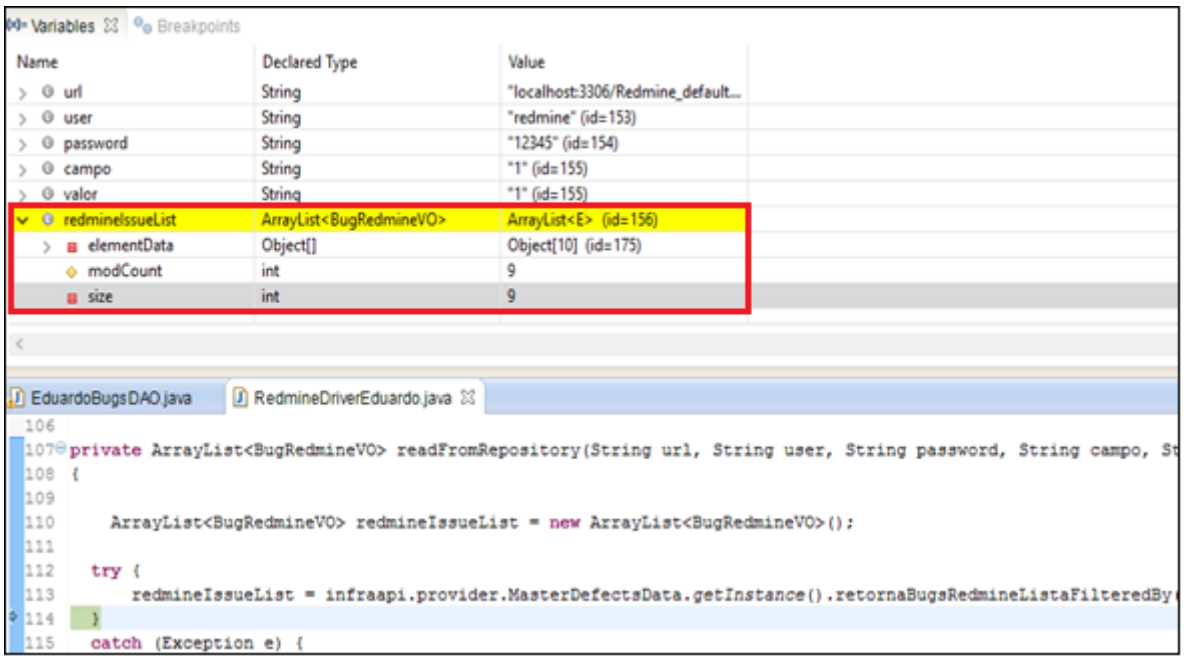


Figura 5.15: Retorno da lista de dados do *Redmine SINAPAD* já no *driver*.

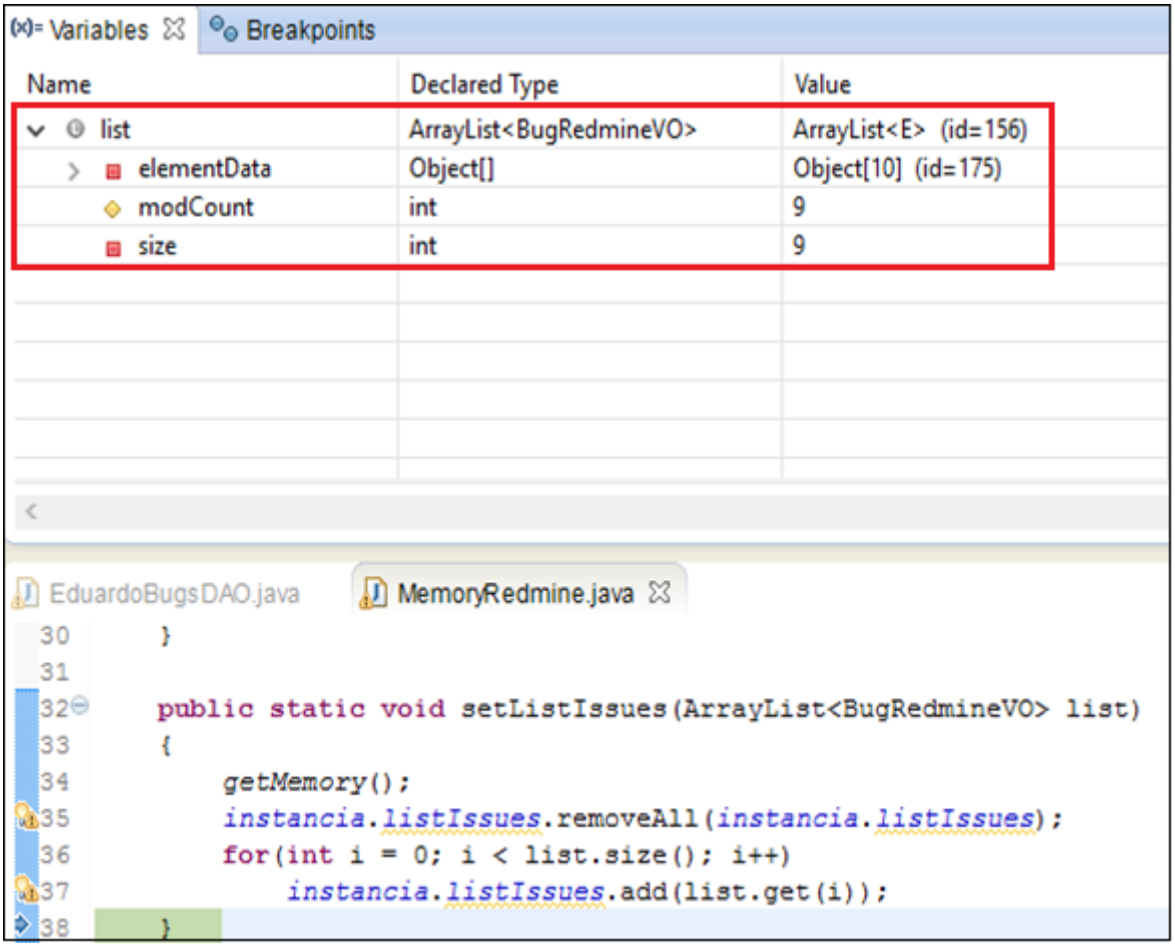


Figura 5.16: Memória de execução da *GiveMe Views* carregada com registros

A execução desse caso de teste também evidenciou que não há o risco de ocorrência

de alteração ou duplicação de dados, entre as etapas de chamada na *InfraAPI*, retorno de informações e carga na memória de execução. Os dados embasarão análises e métricas fiéis ao que, de fato, foi cadastrado na ferramenta de gerenciamento de defeitos do projeto. Ficou claro também que o processo de automatização ocorre em uma única etapa para o usuário final, o que diminui sua complexidade e encurta o processo de análise.

5.3 Avaliação

A prova de conceito realizada foi concebida para demonstrar a efetividade e o fluxo da transmissão dos dados. Por esse motivo, não se fez necessário incluir uma grande massa de dados nos casos de teste, pois poderia ocasionar em falta de clareza no momento de se destacar as evidências representadas pelas imagens.

Pelo mesmo motivo anteriormente mencionado, não se fez necessário executar o *GiveMe Vews* para demonstrar manualmente como são realizadas as análises. Se fez necessário garantir que os mesmos dados persistidos no banco de dados de fato eram iguais aos carregados em memória de execução.

A utilização das informações replicadas do *GCC* e *SINAPAD* representaram uma ilustração para destacar a escalabilidade da *infraAPI*. O *plugin* permite que sejam incluídos novos, e diferentes métodos de integração em sua estrutura modularizada, bastando-se seguir o padrão de projeto.

Como pôde ser constatado, os dez registros de defeitos cadastrados no banco de dados do *Bugzilla GCC* e os outros nove no banco de dados do *Redmine SINAPAD* foram completamente carregados na memória do GiveMe Views, sem que houvesse duplicação ou risco de alteração manual de seus valores. Esse resultado provou sua eficácia e aderência ao que se propôs de realização para este trabalho.

6 Considerações Finais e Trabalhos Futuros

As técnicas de integração, como meio da troca de informações automatizada entre sistemas, auxiliam na construção da ideia de promover componentes cada vez mais modulares. Com funcionalidades únicas e uma capacidade de disponibilizar essas funcionalidades sob forma de serviços, é possível aumentar a reutilização dos produtos de *software*, garantindo benefícios como diminuição da necessidade de grandes estruturas de armazenamento, melhoria da integridade dos dados trocados, aumento da possibilidade de soluções se *software*, que podem ser construídas ao compor módulos distintos e otimização de processamento, pelo fato de utilizar-se somente módulos necessários à resolução de problemas específicos.

A *InfraAPI* oferece esse aspecto de modularidade ao disponibilizar serviços que atendem à necessidade de diferentes ferramentas, que se utilizam de dados advindos de outras bases. Também é eficaz na capacidade de organizar os dados requisitados, representando-os conforme realmente estavam armazenados, minimizando o risco de alterações entre as fases de extração e disponibilização. Sua contribuição ao atender a *GiveMe Infra*, fez com que o processo de utilização dos seus componentes e *plugins* que dependem do *input* de dados históricos fosse beneficiado, trocando informações de maneira direta.

Obteve-se sucesso no mapeamento sistemático realizado como primeiro objetivo desse trabalho, pois foram retornados estudos consistentes e preocupados em resolver as soluções de ausência de modularidade. Foram utilizados alguns conceitos, que já estão sendo implementados neste contexto, para formular o conceito da *InfraAPI*.

O objetivo secundário desse trabalho, que foi a implementação de um mecanismo baseado em técnicas de integração entre sistemas, também foi atingido, conseguindo realizar exportações automáticas de dados de diferentes ferramentas e expondo-os para que fossem consumidos por outras estruturas. A prova de conceito evidenciou o sucesso, principalmente quando testou-se a funcionalidade de recuperação de dados das diferentes bases.

As limitações identificadas na *InfraAPI* se materializam no momento em que

necessita-se de sua funcionalidade fora do ambiente *Eclipse*. Ainda não é capaz de ser uma ferramenta independente dessa plataforma de desenvolvimento, pelo fato de inicialmente ter sido idealizada com este fim. Posteriormente podem ser realizadas modificações para que se torne uma API independente do ambiente de execução.

A prova de conceito também apresentou limitações no momento das execuções nos ambientes em que os repositórios estavam instalados. O fato se deu por motivo de impedimento de acesso direto ao banco de dados dos projetos abertos *GCC* e *SINAPAD*. Houve a necessidade de replicar alguns dados para uma base instalada em ambiente local, representando esses dados reais. A proposta não se prejudicou porque o objetivo principal era verificar como os dados eram movimentados de um ponto ao outro, o foco não era na informação que eles traziam consigo.

Como trabalho futuro, a proposta é tornar a *InfraAPI* cada vez mais independente de sua plataforma sendo capaz de funcionar como um conector genérico baseando-se o padrão *Abstract Factory* que ——————PAREI AQUI —————— , que possibilite a escolha de quaisquer bases a serem integradas de maneira genérica, utilizando , além de ter sua arquitetura modificada para ser formada por componentes com funcionalidades únicas.

Referências Bibliográficas

Ansari, N.; Thampi, G. T. **Template creation to merge disparate software solutions by adapting software engineering principles**. In: 2015 International Conference on Green Computing and Internet of Things (ICGCIoT), 2015.

Apache. **The apache software foundation**. `https://httpd.apache.org/`. Site, acessado 05 mar. 2017.

Barrett, D. J.; Clarke, L. A.; Tarr, P. L. ; Wise, A. E. A framework for event-based software integration. **ACM Trans. Softw. Eng. Methodol.**, 1996.

Blanchette, J. The little manual of api design. **Trolltech, Nokia**, 2008.

Borkar, V. **Liquid data for weblogic: Integrating enterprise data and services**. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data. ACM, 2004.

Bugzilla. **Bugzilla**. `https://www.bugzilla.org/`. Site, acessado 05 mar. 2017.

Carey, M. J.; Haas, L. M.; Schwarz, P. M.; Arya, M.; Cody, W. F.; Fagin, R.; Flickner, M.; Luniewski, A. W.; Niblack, W.; Petkovic, D. ; others. **Towards heterogeneous multimedia information systems: The garlic approach**. In: Research Issues in Data Engineering, 1995: Distributed Object Management, Proceedings. RIDE-DOM'95. Fifth International Workshop on. IEEE, 1995.

Carneiro, G. d. F. Sourceminer: Um ambiente integrado para visualização multiperspectiva de software. 2013.

Councill, B.; Heineman, G. T. Definition of a software component and its elements. **Component-based software engineering: putting the pieces together**, 2001.

Davis, L. A.; Payton, J. ; Gamble, R. **How system architectures impede interoperability**. In: Proceedings of the 2Nd International Workshop on Software and Performance, WOSP '00, 2000.

Draper, D.; Halevy, A. Y. ; Weld, D. S. The nimble integration engine. **SIGMOD Rec.**, 2001.

Foundation, T. E. **Eclipse ide**. `https://eclipse.org/ide/`. Site, acessado 05 mar. 2017.

Free Software Foundation, I. **Gcc, the gnu compiler collection**. `https://gcc.gnu.org/`. Site, acessado 05 mar. 2017.

Geraldo, J. B. Apresentação e aplicação de técnicas de integração de sistemas legados. **Faculdade Governador Ozanam Coelho - FAGOC**, 2015.

Goodall, J. L.; Horsburgh, J. S.; Whiteaker, T. L.; Maidment, D. R. ; Zaslavsky, I. A first approach to web services for the national water information system. **Environmental Modelling & Software**, 2008.

Hamdan, S.; Alramouni, S. A quality framework for software continuous integration. **Procedia Manufacturing**, 2015.

Humphrey, W. S. The software engineering process: Definition and scope. **SIGSOFT Softw. Eng. Notes**, v.14, 1988.

IEEE. **Ieee standard computer dictionary**, 1991.

Oracle, J. **Jdbc**. `http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html`. Site, acessado 05 mar. 2017.

Kitchenham, B. Procedures for performing systematic reviews. **Keele, UK, Keele University**, v.33, p. 2004, 2004.

Lee, S.-H.; Jeong, Y.-S. A system integration framework through development of {ISO} 10303-based product model for steel bridges. **Automation in Construction**, 2006.

Lélis, C. A. S. **Giveme trace: Uma ferramenta de apoio à rastreabilidade de software**. 2014. Dissertação de Mestrado - Universidade Federal de Juiz de Fora.

Mannai, D. N.; Bugrara, K. Enhancing inter-operability and data sharing in medical information systems. **SIGMOD Rec.**, 1993.

Marinos, L.; Papazoglou, M. ; Lee, J. **Using a unifying mediator in an environment of distributed heterogeneous databases**. In: System Sciences, 1991. Proceedings of the Twenty-Fourth Annual Hawaii International Conference on. IEEE, 1991.

Martins, V. M. M. **Integração de sistemas de informação: Perspectivas, normas e abordagens**. 2005. Dissertação de Mestrado - Universidade do Minho Guimarães.

Masseroli, M.; Bonacina, S. ; Pinciroli, F. **Java-based browsing, visualization and processing of heterogeneous medical data from remote repositories**. In: Engineering in Medicine and Biology Society, 2004. IEMBS '04. 26th Annual International Conference of the IEEE. IEEE, 2004.

Mayer, J.; Melzer, I. ; Schweiggert, F. **Lightweight plug-in-based application development**. In: Net. ObjectDays: International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World, p. 87–102. Springer, 2002.

MySQL. **Mysql**. `https://www.mysql.com/`. Site, acessado 05 mar. 2017.

Naidu, P. G.; Palakal, M. J. ; Hartanto, S. **On-the-fly data integration models for biological databases**. In: Proceedings of the 2007 ACM Symposium on Applied Computing. ACM, 2007.

Offutt, A.; Harrold, M. J. ; Kolte, P. A software metric system for module coupling. **Journal of Systems and Software**, 1993.

Orzechowski, T.; Dziech, A. ; Matiolanski, A. **Framework for integration of police repositories**. In: Control and Communications (SIBCON), 2011 International Siberian Conference on. ACM, 2011.

Pang, L.; Zhong, R. Y.; Fang, J. ; Huang, G. Q. Data-source interoperability service for heterogeneous information integration in ubiquitous enterprises. **Advanced Engineering Informatics**, 2015.

Pressman, R. **Engenharia de Software**. McGraw Hill Brasil, 2011.

Redmine. **Redmine**. `http://www.redmine.org/`. Site, acessado 05 mar. 2017.

Rus, I. Model for the integration of software components into heterogeneous it systems. **Procedia Economics and Finance**, v.15, p. 1104, 2014.

SINAPAD. **Sinapad**. `https://www.lncc.br/sinapad/index.php`. Site, acessado 05 mar. 2017.

Seligman, L.; Mork, P.; Halevy, A.; Smith, K.; Carey, M. J.; Chen, K.; Wolf, C.; Madhavan, J.; Kannan, A. ; Burdick, D. **Openii: An open source information integration toolkit**. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. ACM, 2010.

Silva, A. N.; Carneiro, G.; Zanin, R.; Dal Poz, A. ; Martins, E. **Propondo uma arquitetura para ambientes interativos baseados em multiplas visões**. In: II Workshop Brasileiro de Visualização de Software, p. 1–8. UNIFACS, 2012.

Lapes. **Start**. `http://lapes.dc.ufscar.br/tools/start_tool`. Site, acessado 10 mar. 2017.

Steinmacher, I. Chaves, A. G. M. Awareness support in distributed software development: A systematic review and mapping of the literature. **Journal of Computer Supported Cooperative Work**, 2012.

Tavares, J. F.; Braga, R.; David, J. M. N.; Araújo, M. A. P. ; Campos, F. Giveme metrics-um framework conceitual para extração de dados históricos sobre evolução de software. **X Simpósio Brasileiro de Sistemas de Informação (SBSI), Londrina/PR. 2014**, 2014.

Tavares, J. F. **Giveme infra: Uma infraestrutura baseada em múltiplas visões interativas para apoiar a evolução distribuídada de software**. 2015. Dissertação de Mestrado - Universidade Federal de Juiz de Fora.

Tavares, J. F.; Braga, R.; David, J. M. N.; Araújo, M. A. P.; Campos, F. ; Carneiro, G. d. F. Giveme views: uma ferramenta para análise visual de evolução de software a partir de repositórios de dados. 2015.

Thiran, P.; Hainaut, J.-L.; Bodart, S.; Deflorenne, A. ; Hick, J.-M. **Interoperation of independent, heterogeneous and distributed databases. methodology and case support: the interdb approach**. In: Cooperative Information Systems, 1998. Proceedings. 3rd IFCIS International Conference on. ACM, 1998.

W3C. **World wide web consortium (w3c)**. `https://www.w3.org/`. Site, acessado 05 mar. 2017.

WEBrick. **Webrick web server toolkit**. `https://ruby-doc.org/stdlib-2.0.0/libdoc/webrick/rdoc/`. Site, acessado 05 mar. 2017.

Wileden, J. C.; Kaplan, A. **Software interoperability: Principles and practice**. In: Software Engineering, 1997., Proceedings of the 1997 (19th) International Conference on, 1997.

Sherif Yacoub, H. A.; Mili, A. Characterizing a software component. 1999.

Gong, Y.-G.; Chen, X. **Healthcare information integration and shared platform based on service-oriented architectures**. In: Signal Processing Systems (ICSPS), 2010 2nd International Conference on. ACM, 2010.

c. Zhao, C.; y. Zhao, L. **The research about software integration oriented heterogeneous architecture style**. In: The 2nd International Conference on Software Engineering and Data Mining, 2010.