UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ
BACHARELADO EM ENGENHARIA DE SOFTWARE

BRUNO FURTADO CAMPOS

ESTUDO DE CASO DA MIGRAÇÃO DE UM SISTEMA LEGADO PARA ARQUITETURA ORIENTADA A SERVIÇOS

QUIXADÁ
2013

BRUNO FURTADO CAMPOS

ESTUDO DE CASO DA MIGRAÇÃO DE UM SISTEMA LEGADO PARA ARQUITETURA ORIENTADA A SERVIÇOS

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Bacharelado em Engenharia de Software da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.

Área de concentração: computação

Orientador Prof. Camilo Camilo Almendra

QUIXADÁ
2013

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca do Campus de Quixadá

C212e Campos, Bruno Furtado
Estudo de caso da migração de um sistema legado para a arquitetura orientada a serviço / Bruno Furtado Campos – 2013.
50 f. : il. color., enc. ; 30 cm.

Monografia (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2013.
Orientação: Prof. MSc. Camilo Camilo Almendra
Área de concentração: Computação

1. **Arquitetura de computador** 2. Software – controle de qualidade 3. Software - desenvolvimento I. Título.

CDD 004.65

BRUNO FURTADO CAMPOS

ESTUDO DE CASO DA MIGRAÇÃO DE UM SISTEMA LEGADO PARA ARQUITETURA ORIENTADA A SERVIÇOS

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Bacharelado em Engenharia de Software da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.

Área de concentração: computação

Aprovado em: _____ / dezembro / 2013.

BANCA EXAMINADORA

_____
Prof MSc. Camilo Camilo Almendra (Orientador)
Universidade Federal do Ceará-UFC

_____
Prof. Dr. Davi Romero de Vasconcelos
Universidade Federal do Ceará-UFC

_____
Prof. Dr. Lincoln Souza Rocha
Universidade Federal do Ceará-UFC

AGRADECIMENTOS

Aos meus pais, que me deram total apoio em todos os âmbitos para que chegasse até aqui.

Ao meus amigos de graduação, que compartilharam comigo, durante os últimos anos, os mesmos medos, anseios e alegrias, pelo companheirismo.

Ao meus amigos de logos anos, Gleide e Mariana, pela amizade.

Aos professores, Lincoln, Davi e Jeffeson, pelas contribuições dadas à esse trabalho.

Ao professor Márcio, que acreditando em mim, me concedeu a minha primeira oportunidade profissional assumindo reais responsabilidades durante minha graduação.

Ao professor Flávio, que como supervisor de estágio e ex-coordenador de curso, sempre me apoiou.

E ao professor Camilo, que devido as minhas constantes dúvidas e incertezas, serviu de guia e inspiração não só para a conclusão desse trabalho, mas também para a conclusão desse curso. Sem duvidas, sem ele não teria sido possível chegar até aqui.

Um sincero,
Muito Obrigado.

“Nada no mundo se compara à persistência. Nem o talento; não há nada mais comum do que homens malsucedidos e com talento. Nem a genialidade; a existência de gênios não recompensados é quase um provérbio. Nem a educação; o mundo está cheio de negligenciados educados. A persistência e determinação são, por si sós, onipotentes.”
(Calvin Coolidge)

RESUMO

Para que um software continue trazendo vantagens competitivas para as organizações que as utilizam é necessário que ele seja passível de mudanças, contudo em sistemas legados a evolução e a manutenção é uma tarefa complicada. Este trabalho propõe para um sistema legado um processo de migração para a Arquitetura Orientada a Serviços a fim de que se possa tirar proveito dos principais benefícios que essa arquitetura oferece dentre os quais, especificamente, facilitar a criação e o desenvolvimento de novas aplicações utilizando infraestrutura e processos de negócio já existentes. Dessa forma, mantendo o software pronto para responder a mudanças e tornando a organização competitiva dentro da sua área de atuação. A avalição desse trabalho foi realizada a partir do estudo de caso da execução do processo e como principais resultados obtidos tivemos um processo de migração que atendeu com sucesso as expectativas.

Palavras chave: Arquitetura Orientada a Serviço. Sistemas Legados. Migração de sistemas.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Em contabilidade, ativos são “benefícios econômicos futuros prováveis, obtidos ou controlados por uma entidade” (Hendriksen & Van Breda, 1999 apud GOULART, A. M. C, 2002, p. 59). Dessa forma produtos de softwares podem ser considerados como ativos já que, em geral, trazem ou procuram trazer vantagens e benefícios competitivos para as organizações que as utilizam. Contudo, a partir que do momento em que o software deixa de ser evoluído, ele acaba perdendo ou diminuindo essa vantagem, podendo até mesmo chegar a se tornar um problema para as organizações que o utiliza.

Para que o software continue sendo um ativo organizacional, é necessário estar alinhado com os objetivos de negócio e prover sempre a capacidade de mudança para que as organizações possam se manter competitivas dentro do mercado. Assim, as organizações devem estar preparadas para o dinamismo que o mercado exige, devendo estar preparadas para desenvolver sistemas que possam ser extensíveis e facilmente integrados.

Para atender as necessidades das organizações, os sistemas normalmente precisam trocar informações entre diferentes sistemas e se integrar com outras aplicações. Essa integração dos sistemas e das informações é uma questão complexa e desafiadora e que depende de muitos fatores como arquitetura dos sistemas, sistemas operacionais, tipos de componentes, informação a ser integrada, acoplamento e uso dos sistemas, requisitos de desempenho, dados heterogêneos, middleware e interfaces de usuários (CALLE et al, 2012).

Além da complexidade da integração, muito dos sistemas das organizações pertencem a um grupo que costumamos chamar de sistemas legados. Esses sistemas possuem características que dificultam a sua evolução ou manutenção.

Uma das possíveis soluções para tentar resolver o problema da integração de aplicações e que soluciona em parte e de maneira indireta os problemas dos sistemas legados, seria a utilização da Arquitetura Orientada a Serviços (SOA) nas aplicações que armazenam ou lidam com dados ou processos organizacionais importantes.

Segundo Josuttis (2011, p. 22), SOA é um paradigma de arquitetura para lidar com os processos corporativos distribuídos em um grande cenário de sistema heterogêneos existentes e novos que estão sob controle de diferentes proprietários.

A migração de sistemas legados para SOA procura fazer com que funcionalidades legadas sejam expostas para clientes remotos, através da implementação de novos processos de negócio utilizando os ativos de software já existentes ou de terceiros, buscando a redução de custos enquanto aumenta o potencial de inovação através do investimento em software. Além de permitir a descoberta, a composição e a utilização desses serviços para a criação de novas aplicações ou serviços, SOA incentiva o emprego de boas práticas e técnicas de desenvolvimento permitindo o uso de tecnologias atuais que são passiveis de mudança.

Além disso, sistemas legados, em geral, apresentam uma grande quantidade de funcionalidades e que juntamente ao longo período de utilização, acabam gerando um grande volume de dados. Esses dados acabam por sua vez, agregando ainda mais valor ao sistema. Por isso, a necessidade da realização de evoluções na arquitetura se tornam necessárias para que esses sistemas continuem sendo um ativo para a organização.

Contudo, a realização da migração deve ser considerada mais complexa que o processo de desenvolvimento de um novo sistema porque além das dificuldades encontradas normalmente no desenvolvimento de um novo sistema, tais como compreensão dos requisitos, das tecnologias e prazos, é necessária a compreensão total do sistema legado o qual se quer realizar a migração. Isso deve-se às características que cada sistema possui como requisitos funcionais e não funcionais que são únicos e que podem determinar como a migração desses sistemas devem ser realizadas.

Esse trabalho objetivou estabelecer para um sistema legado um processo de migração utilizando SOA para que se pudesse tirar proveito dos principais benefícios que essa arquitetura oferece os quais, especificamente, facilitar a criação e o desenvolvimento de novas aplicações utilizando infraestrutura e processos de negócio já existentes. A fim de validar o processo de migração proposto, um estudo de caso de evolução de sistema legado foi realizado. Esse trabalho relata esse estudo de caso e faz uma avaliação do processo de migração.

Para facilitar a leitura, esse trabalho está organizado em sete capítulos. No capítulo 2 são apresentadas as referências bibliográficas necessárias para a compreensão desse trabalho. No capítulo 3 são apresentados os procedimentos metodológicos. No capítulo 4, o sistema legado alvo desse trabalho é apresentado e características do sistema são discutidas. No capítulo 5, é apresentado detalhadamente o processo de migração que foi criado especificamente para esse trabalho como forma de atender as necessidades da organização e da aplicação. No capítulo 6, iremos relatar como se deu a identificação dos ativos, justificando

as escolhas tomadas, tanto de design como de tecnologias, relatando as dificuldades encontradas, as lições aprendidas e as oportunidades de melhorias. No capítulo 7, iremos realizar uma avaliação dos serviços criados procurando assim validar a ideia a qual esse trabalho se objetiva. Por fim, no capítulo 8, apresentaremos a conclusão, apresentando também possíveis propostas para trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

Nessa seção serão explicados, definidos e comparados os benefícios, métodos e conceitos que serão utilizados por esse trabalho como forma de introduzir ao leitor uma maior compreensão sobre a área a qual será aqui discutida.

2.1 Arquitetura Orientada a Serviços

Josuttis (2011, p. 22) define SOA como "um paradigma de arquitetura para lidar com os processos corporativos distribuídos em um grande cenário de sistemas heterogêneos existentes e novos que estão sob controle de diferentes proprietários". Já Kontogiannis, Lewis e Smith (2008) afirmam que existem duas definições para SOA dependendo da perspectiva escolhida: a técnica ou a de negócios.

Na perspectiva técnica, SOA é definido como uma abordagem para desenvolvimento de software na qual os serviços disponibilizam funcionalidades reutilizáveis através de interfaces bem definidas, além de contar com uma infraestrutura que permite a descoberta, a composição e a invocação (utilização) desses serviços, permitindo a criação de novas aplicações que utilizem esses serviços.

Na perspectiva de negócio, SOA é definido como uma forma de expor funcionalidades legadas para aplicações clientes remotas, implementando novos processos de negócio utilizando os ativos de software existentes ou de terceiros, reduzindo os custos enquanto aumenta o potencial de inovação através do investimento em software.

O aspecto de negócio é um ponto importante para o sucesso de SOA nas organizações, dessa forma Kontogiannis, Lewis e Smith complementam a definição de Josuttis, focando sua definição nos dois aspectos e deixando claro a existência de ambos, diferentemente de Josuttis que não explicita isso.

Prosseguindo, Newcomer e Lomow (2004) definem SOA como um estilo de modelagem orientado a todos os aspectos de criação e utilização de serviços corporativos por todo o seu ciclo de vida (desde a criação até a aposentadoria), assim como de definição e fornecimento de uma infraestrutura de TI que permita que as aplicações diferentes troquem informações e participem de processos corporativos, independentemente de sistemas operacionais ou linguagens de programação por detrás dessas aplicações.

Newcomer e Lomow possuem uma definição mais completa em que conseguem agregar em um único conceito ambas as características, técnicas e de negócio, de SOA. Percebe-se pelas definições anteriores que SOA não é um framework ou tecnologia específica e sim um estilo, paradigma, conceito, perspectiva, filosofia que guiará a construção de aplicações concretas.

Surgida para resolver as necessidades que as organizações possuem, SOA sugere disponibilizar funcionalidades especificas de seus sistemas para clientes e parceiros e com isso integrar outras aplicações que as utilizam. Contudo, para que isso seja realizado é necessário que os sistemas desenvolvidos sejam flexíveis, interoperáveis e possuam um baixo acoplamento (JOSUTTIS, 2008). Tais características são princípios dos sistemas baseados nesta arquitetura.

SOA baseia-se nos seguintes princípios de design de serviços abaixo (ERL, 2009):

- Serviços são reutilizáveis.
- Serviços compartilham um contrato formal.
- Serviços possuem um baixo acoplamento.
- Serviços abstraem a lógica.
- Serviços são capazes de se comporem.
- Serviços são autônomos.
- Serviços evitam alocação de recursos por longos períodos.
- Serviços são capazes de serem descobertos.

Esses princípios também podem ser considerados como as suas principais vantagens e a sua utilização em uma arquitetura que as implementam permitem, por exemplo que diferentes sistemas – independentemente de sistema operacional ou linguagem de programação utilizada – comuniquem-se e troquem informações sem a necessidade de conhecer todo o sistema a qual se que integrar.

Uma outra vantagem da utilização dessa arquitetura é a possibilidade de integração com sistemas legados através do compartilhamento de um contrato formal, como prega os princípios de SOA. Sistemas legados são definidos como qualquer sistema de informação que resiste de forma significativa a modificação e evolução (BISBAL et al,1999). Dificuldades tais, por exemplo referentes a inviabilidades técnicas (ex: linguagem ou

framework de desenvolvimento obsoleta), econômicas (ex: custo muito elevado) ou de negócio (ex: sistemas críticos que não podem ser interrompidos). Esses sistemas legados costumam possuir um custo de manutenção elevado para a organização.

A utilização de migração ou implantação de aspectos da arquitetura orientada a serviços nesses sistemas permite a reutilização da aplicação legada utilizando funcionalidades já existentes, provendo apenas um contrato de utilização permitindo acesso de outros clientes ao serviço.

O investimento na migração de sistemas para a arquitetura de serviços permite às empresas, além de obter todas as vantagens que essa arquitetura oferece, a possibilidade de continuar fazer valer o investimento inicial do software em vez de simplesmente descartá-lo. Além disso, os sistemas legados podem ser considerados artefatos importantes para a organização, já que ali foi investido tempo, dinheiro e expertise de vários usuários garantindo vantagens competitivas dentro do mercado a qual ela faz parte.

Porém, SOA não é adequada para resolver todos os problemas, sua utilização no entanto, é um processo complexo não somente em aspectos técnicos, mas também em aspectos de negócio. Isso por que a implantação e manutenção de sistemas que utilizam essa arquitetura exigem um sistema gerenciamento e manutenção continua, devendo essa decisão ser tomada juntamente com a gerência da organização para que se possa obter seu apoio garantindo assim sua manutenção de forma continua e longínqua.

É importante destacar ainda que não existe uma abordagem de migração padrão ou que se adeque a todos os casos e sistemas. As organização e seus sistemas possuem características próprias que determinam como a migração deverá ser realizada. Técnicas e práticas da indústria e da academia costumam ser definidas no processo para serem utilizadas na migração e as suas e escolhas são através da análise das características do sistema, do ambiente organizacional e das pessoas envolvidas verificando sempre se a elas se adequação contexto da aplicação e da migração.

2.2 Evolução de Software

Para que um software continue sendo um ativo organizacional importante, o investimento sobre ele realizado seja válido e que traga de fato vantagens competitivas no mercado, é necessário então saber quando um sistema necessita ou não de mudança e se elas valem apena. Sommerville (2011) descreve quatro casos para que isso possa ser identificado:

a) Acabar com o sistema totalmente:

   Esta opção deve ser escolhida quando o sistema não está contribuindo de maneira efetiva para os processos de negócios da organização. Isso geralmente ocorre quando os processos de negócios mudaram desde que o sistema foi instalado e não são mais dependentes do sistema legado.

b) Deixar o sistema inalterado e continuar com a manutenção regular:

   Esta opção deve ser escolhida quando o sistema ainda é necessário, mas é bastante estável e os usuários do sistema fazem poucas solicitações de mudança.

c) Reestruturar o sistema para melhorar a sua capacidade de manutenção:

   Esta opção deve ser escolhida quando o sistema de qualidade tem sido degradada pelas mudanças e onde ainda está sendo proposta novas mudanças para o sistema. Este processo pode incluir em desenvolvimento de novos componentes de interface para que o sistema original possa trabalhar com outros sistemas mais recentes.

d) Substituição total ou parcial do sistema com um novo sistema:

   Essa opção deve ser escolhida quando fatores como um novo hardware, implica que um sistema antigo não pode continuar em operação ou aonde módulos de sistemas de prateleira permitiram o desenvolvimento de um novo sistema com um custo razoável. Em muitos casos, a estratégia de substituição evolutiva pode ser adotada como principal forma de desenvolvimento, aonde módulos do sistema de substituídos por módulos de sistemas de prateleira com reutilizando componentes existentes sempre que possível.

Caso haja de fato a necessidade de mudança ou seja, se os itens **c)** e **d)** forem aplicáveis, será necessário verificar qual abordagem seguir em relação ao desenvolvimento do novo sistema. Existem três estratégias para lidar com sistemas legados: *Wrapping* (envolvimento), *Redevelopment* (Re-desenvolvimento) e *Migration* (Migração) (BISBAL et al,1999).

Na primeira estratégia, *Wrapping* ou envolvimento, o sistema legado é envolvido como um componente de caixa preta onde outro sistema mais estruturado será desenvolvido

para que possa reutilizar as funcionalidades existentes do sistema legado, garantindo a total funcionalidade e compatibilidade do sistema sem comprometer seu funcionamento.

Na segunda estratégia, *Re-development* ou Re-desenvolvimento, o sistema legado é abandonado totalmente e um novo sistema será desenvolvido e projetado com novas tecnologias e melhores práticas para substituir totalmente o sistema legado, reimplementando as funcionalidades do sistema anterior além de permitir a extensão e realização da manutenção de forma simplificada.

A terceira estratégia, *Migration* ou migração, é um meio termo entre as duas abordagens anteriores. Ela procura desenvolver um novo sistema em paralelo com a utilização do sistema legado existente. Um exemplo seria desenvolver uma funcionalidade do sistema legado em um novo sistema, utilizando uma nova tecnologia e desabilitar o acesso a essa funcionalidade no sistema legado, fazendo com que os usuários a utilizassem a funcionalidade do novo sistema. Dessa forma, o sistema seria migrado aos poucos, garantindo a corretude do novo sistema.

2.3 Processo de migração para SOA

O processo de migração de um sistema qualquer é algo complexo que demanda tempo e experiência para ser executado com sucesso. Isso se deve a diversos fatores como tipo de software a ser migrado e aspectos operacionais da organização. Além disso, a realização de migração de sistemas para arquitetura SOA é algo que aumenta a complexidade. Isso por que nesse tipo de migração aspectos organizacionais são ainda mais impactantes para o comportamento da migração e do funcionamento do sistema.

Um sistema difícil de ser migrado, por exemplo, é um sistema de grande porte com vários módulos e que possui uma grande base de dados de usuários e que os mesmos, podem possuir resistência a mudança. Nesse caso hipotético, uma alternativa possível seria a migração dos módulos de forma gradual, fazendo com que os usuários fossem treinados e se acostumassem com o novo sistema.

Outro caso hipotético de migração, agora considerando que a migração será realizada para a arquitetura SOA e levando em consideração que esse tipo de migração pode alterar ou não os processos da organização, seria um no qual uma empresa com várias filiais, aonde cada filial teria seu próprio software de gerenciamento e os relatórios mensais que devem ser enviados de forma impressa para a matriz, agora serão enviados digitalmente. Nesse caso migração procuraria criar serviços nesses softwares aonde haveriam serviços que

liberariam uma interface de acesso especifica para esse processo para que assim a matriz possa receber esses relatórios e trata-los da forma que achar necessário.

Tendo em vista as dificuldades de migração de sistemas tanto comum quanto para SOA e as diferentes formas de construir um processo de migração, existem trabalhos que explicam técnicas e boas práticas de como realizar a migração além de ajudar a evidenciar funcionalidades que devem ser migradas para atender as necessidade das organizações no sentido de integrar as aplicações e trabalhos que relatam a experiência de migração de sistemas e seus respectivos processos de migração.

Um exemplo de trabalho realizado para facilitar o entendimento do processo de migração e da realização da própria migração em si é o framework conceitual chamado SOA-MF definido por Razavian e Lago (2010) que serve de guia para a migração de sistemas legados para SOA. O framework explica a existência de fases especificas como reengenharia, transformação e desenvolvimento aonde cada uma representa um momento importante para o entendimento e a migração do sistema.

Primeiramente, na fase de reengenharia, o sistema legado será analisado desde o código e a arquitetura do sistema, até mesmo as regras de negócio, para que com isso o conhecimento adquirido nessa fase permita transforma-lo em representações de mais alto nível para que possam ser utilizadas nas fases seguintes. Já nas fases seguintes, transformação e desenvolvimento, serão realizados a reestruturação/redesign da representações criadas na fase anterior para que possam ser utilizadas na durante a fase do desenvolvimento.

Já outro artigo também de Razavien e Lago (2011) realiza uma pesquisa elencando um conjunto de técnicas de migração de sistemas para SOA procurando comparar as diferenças entre as abordagens da indústria e da academia, recomendando quais devem ser seguidas durante a migração de sistemas. O estudo concluiu que:

- Diferentes empresas compartilham o mesmo conjunto de atividades para realização da migração.
- A migração realizada pela indústria converge para uma estratégia de migração comum, a estratégia de *migration*.
- A migração realizada pela indústria não utiliza técnicas de engenharia reversa para compreender os sistemas legados.

- O conhecimento necessário é elicitado a partir dos *stakeholders* que conhecem e entendem o sistema.

Além disso, os resultados mostram também que as abordagens de migração da academia não valem o custo-benefício e que é necessário avaliar o sistema legado a partir de múltiplas perspectivas. Procuram também, mostrar a lacuna entre o que é descrito na teoria e o que de fato é utilizado na prática, para comprovar que o que é costumeiramente utilizado na indústria não condiz com o que é realizado meio acadêmico. Finalizam seu trabalho afirmando a necessidade de adequação dos estudos da academia com é utilizado na indústria.

Já Almonaies et al (2011), descrevem um relato de experiência de migração de sistemas para SOA e compartilham as principais dificuldade encontradas, dentre elas está, por exemplo, a identificação de quais serviços ou conjuntos de serviços devem ser migrados para atender ou facilitar alcançar os objetivos de negócio das organizações. Além disso, também é descrito o processo de migração e de identificação de funcionalidades a serem migradas além de realizar um levantamento de casos de uso das funcionalidades que foram migradas.

Outros trabalhos procuram mostrar técnicas e processos mais detalhados para ajudar na identificação de funcionalidades que possam ser migradas para SOA. Como as técnicas de identificação SMART (*Service-Oriented Migration and Reuse Technique*) de Lewis et al (2005), que possui um conjunto mais especifico de técnicas que ajudam na identificação de funcionalidades e além de guias à serem seguidos na hora de realizar uma migração.

Diferentemente do SOA-MF, que indica somente um conjunto de passos e atividades mais genéricas, o SMART, indica de forma mais detalhada o que fazer em cada atividade como, por exemplo, quais stakeholders devem ser entrevistados primeiros e quais os mais importantes.

Arcelli, Tosi e Zanoni (2008) descrevem outro método de identificação de serviços através do uso de *design pattern dectection*, DPD, detecção do uso de padrões de projeto em português. Esse método procura analisar o código legado da aplicação para identificar padrões de projetos que sejam possíveis candidatos a se tornarem serviços devido a sua comportamental ou estrutural.

2.4 Tipos de Serviços

Erl (2008) descreve três classificações básicas referentes à forma como os serviços criados são expostos:

- Serviço de Entidade:

  São serviços que disponibilizam as entidades de negócio da organização, como por exemplo funcionários, clientes, faturas, etc. São considerados altamente reusáveis, devido a sua natureza agnóstica. Também são conhecidos como “serviços de negócios centrados na entidade” ou “serviços de entidade de negócio”.

- Serviço de Tarefa

  São serviços de negócio com limite funcional associado diretamente a uma tarefa ou processo específico da organização. Esse tipo de serviço tende a ser menos reutilizável já que tende a ser mais especifico. Um exemplo desse tipo serviço seria um serviço de prestação de contas entre filiais e a matriz de uma organização, realizado mensalmente, aonde os dados de várias entidades são enviados/requisitados.

- Serviço Utilitário

  Serviços utilitários são dedicados a fornecer funcionalidades reusáveis de serviços, como por exemplo, registro de eventos em log e notificação. Esse são conhecidos como “serviços de infraestrutura” ou “serviços de tecnologia”. Um serviço de conversão de formatos de arquivos, é um exemplo de serviço utilitário.

Apesar de serem diferentes, esses serviços podem ser utilizados em conjunto, convivendo mutualmente em uma mesma aplicação.

3 PROCEDIMENTOS METODOLÓGICOS

Nessa seção serão explicados os passos que foram executados para atingir o objetivo desse trabalho.

a) Definir um processo de migração

   Nessa fase um processo de migração foi definido para ser utilizado na fase migração desse projeto. O processo foi definido tendo como base as características do sistema a ser migrado e de técnicas e boas práticas documentadas pela indústria e pela academia, além de levar em consideração os aspectos funcionais da organização.

b) Análise de tecnologias que serão utilizadas na migração

   Uma análise das tecnologias atuais de desenvolvimento de software, integração de sistemas e SOA foi realizada para escolha de quais tecnologias serão utilizadas durante a migração. A escolha foi feita através da comparação das tecnologias atuais nesses campos verificando aspectos como suporte da comunidade, ferramentas disponíveis, tecnologias *open-source* e etc.

c) Análise de Novas Funcionalidades

   Após a análise das tecnologias, foi necessário identificar as funcionalidades necessárias ao sistema legado que deveriam ser migradas utilizando a nova arquitetura.

d) Executar migração

   Após o processo e as tecnologias a serem utilizadas serem definidas, a migração foi iniciada. Nessa fase, o novo serviço foi criado conforme descrito no processo.

e) Projetar e desenvolver uma nova aplicação que utilize os serviços desenvolvidos

   Uma nova aplicação que utilize os serviços do sistema que foi migrado foi projetada e desenvolvida para que se pudesse validar a facilidade de integração de sistemas com SOA e assim validar se o objetivo do trabalho foi alcançado.

f) Relatar o processo de migração

   Um relato da migração foi escrito abordando todos os obstáculos encontrados.

g) Fazer uma avaliação sobre os serviços criados para validação

   Para concluir, uma discussão foi levantada para demostrar como os serviços criados facilitaram o desenvolvimento das aplicações criadas e de futuras aplicações a serem desenvolvidas.

4 O SISTEMA LEGADO

O Sistema de Presenças e Planos de Aulas (SIPPA) apresenta como funcionalidades principais o controle das presenças dos alunos nas disciplinas nas quais estão matriculados e controle de notas, além disso apresenta um conjunto de funcionalidades para ajudam no controle atividades relacionadas ao ensino e a gestão dos alunos e das disciplinas.

Utilizado principalmente por membros do Campus de Quixadá da Universidade Federal do Ceará, sendo eles alunos, professores, coordenadores, direção e servidores técnico-administrativos, o SIPPA prove uma forma de facilitar a utilização dos processos organizacionais da Universidade, tornando-se uma ferramenta importante para os membros que a compõem.

O sistema em questão apresenta muitas funcionalidades para o gerenciamento de alguns processos organizacionais. Dentre os quais podemos citar: Gerenciamento de alunos, coordenadores, cursos, grade curricular, disciplinas, planos de aulas, turmas, frequência dos alunos, avaliações dos alunos, envio de trabalhos, dentre outras.

Consequentemente devido à grande quantidade de funcionalidades que o sistema apresenta e ao tempo de utilização, um grande volume de informações foram gerados e armazenados. Essas informações acabam agregando ainda mais valor ao sistema, por isso a necessidade da realização de melhoramentos e evoluções na arquitetura se tornam necessárias que o SIPPA continue sendo um ativo para a organização.

O SIPPA que é um sistema web desenvolvido utilizando Servlets e JSP da linguagem Java, não possui uma documentação e também não faz uso de frameworks, contudo faz o uso de alguns padrões de projeto para auxiliar o desenvolvimento.

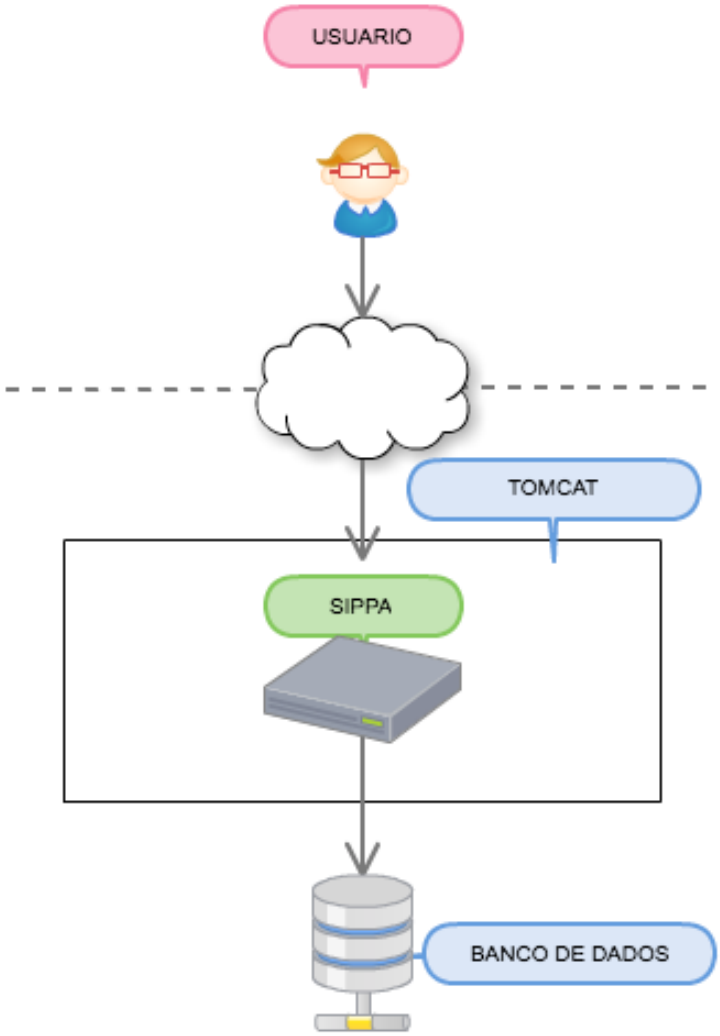

Figura 1 - SIPPA Atualmente

Contudo devido a uma rápida evolução sem um planejamento adequado, fez com que a arquitetura atual não mais se adeque as necessidade da aplicação, fazendo com que as evoluções sejam cada vez mais difíceis e custosas.

5 DEFINIÇÃO DO PROCESSO DE MIGRAÇÃO

Devido à não existência de uma abordagem de migração de sistemas padrão iremos, primeiramente iremos definir um processo de que será utilizado para a migração do SIPPA para a arquitetura SOA, baseando-se nos trabalhos relacionados já descritos.

A definição do processo de migração levou em consideração qual estratégia de migração seria a mais adequada, dentro os tipos *wrapping* , *re-development* ou *migration* (BISBAL et al,1999). A estratégia *migration* foi identificada como mais adequada, em especial por dois fatores. O primeiro pelo grande conjunto de funcionalidades que o SIPPA apresenta, sendo assim, inviável a mudança de todo o sistema para SOA de uma única vez. Já o segundo fator, foi a necessidade de migração de um conjunto especifico de funcionalidades sem substituir o funcionamento do sistema anterior. Dessa forma, as técnicas de *wrapping* e *re-development* não se adequam às necessidades desse projeto.

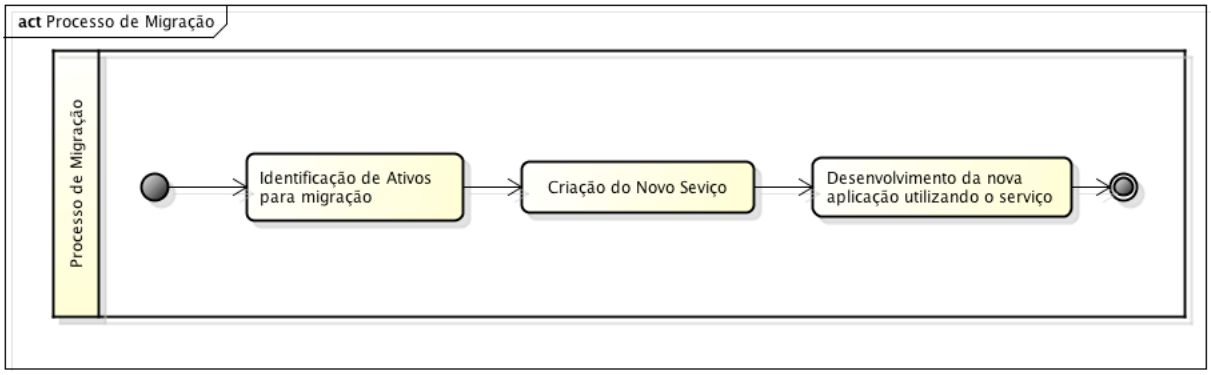

Figura 2 – Processo de Migração

O processo de migração definido está ilustrado na Figura 2. Abaixo seguem a descrição e justificativa de cada fase do processo:

a) Identificação de ativos para migração:

   O processo de migração inicia-se a partir de novos requisitos demandados pelos *stakeholders*, implicam em uma mudança no sistema legado. Dessa forma, nessa fase as funcionalidades legadas que são necessárias para a realização da implementação dos novos requisitos serão analisadas para verificar o impacto da migração. Esse passo é importante, por exemplo, para definir se o sistema já possui comportamento legado que possa ser reusado ou ainda se pré-requisitos das funcionalidade que será migrada, deverá ou não ser migrada também.

b) Criação do novo serviço.

Após a seleção das funcionalidades legadas a serem migradas, a migração será iniciada. O primeiro passo será criar um novo serviço independente do sistema legado (SIPPA), desenhado a partir dos ativos identificados e projetado com características de um Serviço (ERL, 2009). Em seguida, os serviços do SIPPA serão desenvolvidos incluindo toda a lógica e as regras de negócio da funcionalidade. Esse serviço irá ser responsável por controlar o acesso ao banco de dados da aplicação, possuir a lógica de negócio e prover interfaces de acesso (Serviços) para as outras aplicações.

c) Desenvolvimento dos novos requisitos

Após a criação do novo serviço, os novos requisitos que foram o gatilho do processo de migração podem ser desenvolvidos, usando o novo serviço como referência.

Para facilitar o entendimento, a Figura 1 ilustra como SIPPA está atualmente e a Figura 3 como ele irá ficar após a migração das funcionalidades. Já a Figura 4 mostra como ficariam os sistemas após a integração com as novas aplicações.

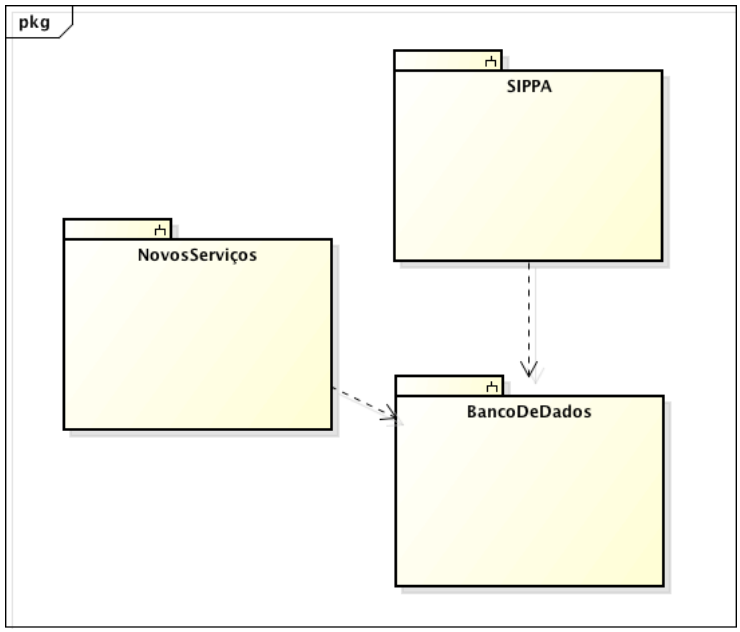

Figura 3 – SIPPA após o processo de migração.

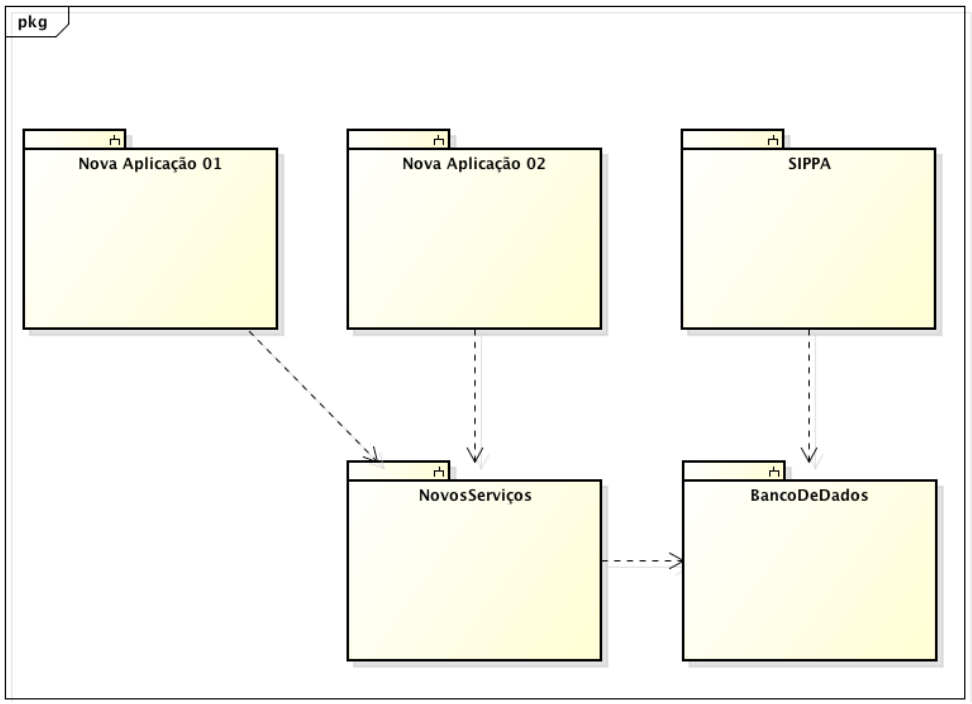

Figura 4 – SIPPA após migração e com outras aplicações integradas.

6 ESTUDO DE CASO: MIGRAÇÃO DO SISTEMA

Com a definição do processo concluída, a migração foi iniciada. A migração compreendeu o desenvolvimento de um conjunto de novas funcionalidades no sistema, que ao invés de ser simplesmente implementado diretamente na base de código atual do sistema legado, foi implementado de acordo com o processo de migração proposto nesse trabalho.

6.1 Nova aplicação

Elaboramos juntamente com os *stakeholders* do sistema legado uma aplicação de apoio para automatizar alguns processos existentes da organização. Nesse caso em questão, o processo de Demanda no qual participam alunos e coordenadores de curso como envolvidos.

No processo de Demanda, as coordenações de curso da UFC são responsáveis pela elaboração das solicitações de disciplinas semestralmente, sendo essa solicitação conhecida como Demanda. As demandas devem considerar a integralização curricular dos cursos, mas podem ser enriquecidas com sugestões dos alunos de disciplinas de maior interesse. A aplicação planejada seria uma aplicação de Pré-Demanda que seria de apoio a decisão, auxiliando os coordenadores na coleta dos interesses dos alunos para o semestre seguinte, ajudando na elaboração na Demanda semestral.

Como funcionalidades principais teríamos:

- Coordenação cria Pré-Demanda.

  O usuário que tem acesso a nível de coordenação irá criar a Pré-Demanda para um semestre especifico selecionando as disciplinas que estão disponíveis para os alunos escolherem e a data de termino para o solicitação da Pré-Demanda.

- Aluno responde a Pré-Demanda.

  Aluno ao acessar a ferramenta irá visualizar a solicitação de Pré-Demanda disponível. Ao escolher a Pré-Demanda a ser solicitada uma lista com as disciplinas que está disponíveis será exibida e assim será possível ao aluno escolher quais as disciplinas ele deseja cursar. Quando a solicitação for concluída uma mensagem de sucesso será exibida.

- Coordenação visualiza resultado da Pré-Demanda.

A coordenação poderá visualizar o resultado da Pré-Demanda que irá conter quais e quantos alunos solicitaram, quais as cadeiras que foram solicitadas e as suas respectivas quantidades de solicitações.

Agora, com os requisitos da nova aplicação levantados, o processo de migração é iniciado. A primeira fase do processo é a identificação dos ativos que foram necessários para o desenvolvimento.

Analisando as funcionalidades da aplicação de pré-demanda identificamos no sistema legado as entidades Aluno, Coordenador, Curso, Disciplina e Pessoa como entidades de negócio necessárias para o desenvolvimento da nova aplicação.

6.2 Avaliação das tecnologias

O primeiro passo que realizamos antes da migração, foi a seleção das tecnologias que seriam utilizadas.

Como linguagem de programação utilizada para o desenvolvimento dos serviços, a linguagem escolhida foi Java, devido a dois motivos: primeiro foi a possibilidade de que pudesse haver alguma oportunidade de reuso do código e o segundo foi o conhecimento que o autor já possuía com a tecnologia em questão.

Posteriormente foi necessário definir qual arquitetura, tecnologia ou estratégia seria utilizada para a integração da aplicação. Devido a grande quantidade de maneiras de implementar serviços, como por exemplo, DCOM, CORBA, SOAP/WSDL, REST, etc, também foi necessário realizar a escolha de qual tecnologia seria utilizada. Contudo a nossa escolha limitou-se a Web Services, que são serviços que baseiam-se no protocolo HTTP (*Hypertext Transfer Protocol*), excluindo, por exemplo, dentre os exemplos citados anteriormente, CORBA e DCOM que funcionam através de RPC (*Remote Procedure Call*).

As principais vantagens da utilização de tecnologias que se baseiam no protocolo HTTP, segundo Daigneau (2011), são por exemplo, a facilidade de reuso e compartilhamento de lógica comum com diversos clientes, devendo-se isso pelo fato do que o protocolo ser um padrão aberto, interoperável e independente de tecnologias de execução.

Nesse trabalho, a escolha da utilização somente em tecnologias *open-source* deu-se em considerações a fatores tais como: quantidade de fornecedores, comunidades de desenvolvimento ativas e código-fonte aberto. A utilização de padrões proprietários não foi

considerada em virtude da possibilidade de ficar preso a único fornecedor o que no futuro poderia tornar o serviço não mais reutilizável violando assim um dos princípios de SOA.

Baseado no protocolo HTTP, temos duas alternativas: REST (*REpresentational State Transfer*) e WSDL (*Web Services Description Language*). Pautasso, Zimmermann e Leymann (2008) descrevem REST como um estilo arquitetural para construção de sistemas distribuídos de grande escala de hipermídia e que possui quatro princípios básicos:

- Interface padrão.
- As mensagens são auto-descritas.
- Interações sem armazenamento de estado (stateless).
- Recursos (Serviços) são identificados através de uma URI única.

Já o WSDL (*Web Service Description Language*) é uma linguagem utilizada para descrever serviços web utilizando o protocolo SOAP. Esse por sua vez, define a arquitetura e o formato das mensagens que serão utilizados no transporte das informações. O WSDL tem como grandes vantagens a promessa de interoperabilidade sendo um protocolo transparente e independente, permite o encapsulamento de algumas tarefas dos desenvolvedor, como *marshalling* (a.k.a serialização), segurança, *state-ful*, permite a utilização de mensagens síncronas e assíncronas, tudo isso, atribuindo responsabilidade a linguagem.

Com isso, o uso de REST foi escolhido, baseando-se no fato de tratar de um padrão aberto, que possui uma simplicidade na manipulação dos dados e ser leve. O uso do WDSL foi descartado, pois as múltiplas implementações disponíveis acabam violam a especificação e fazendo que uma das suas grandes vantagens, a interoperabilidade, seja comprometida. Também possui uma grande quantidade de especificações, que o tornam um padrão burocrático e pesado.

Além disso, outras características arquiteturais de ambas as tecnologias, expostas nas Tabela 1 e 2 por Pautasso, Zimmermann e Leymann (2008) também influenciaram nessa decisão.

Architectural Principle and Aspects	REST	WS-*
Protocol Layering	yes	yes
HTTP as application-level protocol	✓	
HTTP as transport-level protocol		✓
Dealing with Heterogeneity	yes	yes
Browser Wars	✓	
Enterprise Computing Middleware		✓
Loose Coupling, aspects covered	yes, 2	yes, 3
Time/Availability		✓
Location (Dynamic Late Binding)	(✓)	✓
Service Evolution:		
Uniform Interface	✓	
XML Extensibility	✓	✓
Total Principles Supported	**3**	**3**

Tabela 1 - Princípios de Comparação

Architectural Decision and AAs	REST	WS-*
Integration Style	1 AA	2 AAs
Shared Database		
File Transfer		
Remote Procedure Call	✓	✓
Messaging		✓
Contract Design	1 AA	2 AAs
Contract-first		✓
Contract-last		✓
Contract-less	✓	
Resource Identification	1 AA	n/a
Do-it-yourself	✓	
URI Design	2 AA	n/a
"Nice" URI scheme	✓	
No URI scheme	✓	
Resource Interaction Semantics	2 AAs	n/a
Lo-REST (`POST`, `GET` only)	✓	
Hi-REST (4 verbs)	✓	
Resource Relationships	1 AA	n/a
Do-it-yourself	✓	
Data Representation/Modeling	1 AA	1 AA
XML Schema	(✓)[a]	✓
Do-it-yourself	✓	
Message Exchange Patterns	1 AA	2 AAs
Request-Response	✓	✓
One-Way		✓
Service Operations Enumeration	n/a	≥3 AAs
By functional domain		✓
By non-functional properties and QoS		✓
By organizational criterion (versioning)		✓
Total Number of Decisions, AAs	**8**, 10	**5**, ≥10

Tabela 2 - Comparação Conceitual

Para auxiliar o desenvolvimento do serviço também foi utilizado o framework vRaptor que também é desenvolvido em Java suportando as arquiteturas REST e ActionBased, além possuir alguns recursos interessantes como CoC (*Convention over Configuration*), MVC (*Model-view Controoler*), IOC (*Inversion of Control*) e AOP (*Aspect-orinted programming*) que são boas práticas de engenharia.

6.3 Projeto dos serviços

Posteriormente, foi necessário definir como esses serviços seriam expostos. Baseando-se nos tipos de serviços disponíveis, descritos na Seção 2.4 , verificamos que a alternativa mais adequada seria utilização de serviços focados nas entidades pois iria prover uma maior capacidade no aumento do reuso dos serviços criados.

Posteriormente, foi definido as URIs de acesso aos serviços. Na arquitetura REST, para a realização do acesso aos recursos, são utilizados os métodos do protocolo HTTP que são GET, POST, DELETE e PUT. Dessa forma o protocolo acaba se tornando, não apenas uma camada de transporte das informações, mas acaba também fazendo parte da camada de aplicação. Dessa forma, o acesso aos recursos está altamente acoplado com a forma que a requisição é realizada.

Devido às várias implementações do navegadores e versões do HTML, alguns problemas podem surgir, como por exemplo a não implementação de alguns métodos mais recentes como o DELETE e o PUT. Algumas estratégias para evitar são utilizadas, como por exemplo no framework Ruby on Rails todas as requisições utilizam GET ou POST e o real método da requisição é informado como um atributo da requisição.

Contudo para evitar esse tipo de comportamento, definimos nas nossas URIs o método especifico da operação. A estrutura geral das URIs para acesso ficou: http://endereco/entidade/metodo , indicando a entidade e os métodos passíveis de invocação. Por exemplo, para adicionar um coordenador, a URI de acesso seria http://endereco/coordenador/adicionar. Dessa forma, atendemos um dos princípios de REST, que é possuir URI única para um recurso e evitamos problemas com as implementações dos navegadores, versões do HTML e versões do protocolo HTTP.

O próximo passo foi uma análise mais detalhada na aplicação legada para entender como o sistema funciona. Para auxiliar nessa atividade, utilizamos as ferramentas CodePro Analitycs[1] e o ObjetctAid[2] para a visualização de gráficos e estatísticas sobres o projeto.

Uma primeira análise verificou-se um alto grau de acoplamento entre os pacotes do sistema, conforme mostra a Figura 5.

---

1 CodePro Analitycs: https://developers.google.com/java-dev-tools/codepro/doc/

2 ObjectAid: http://www.objectaid.com

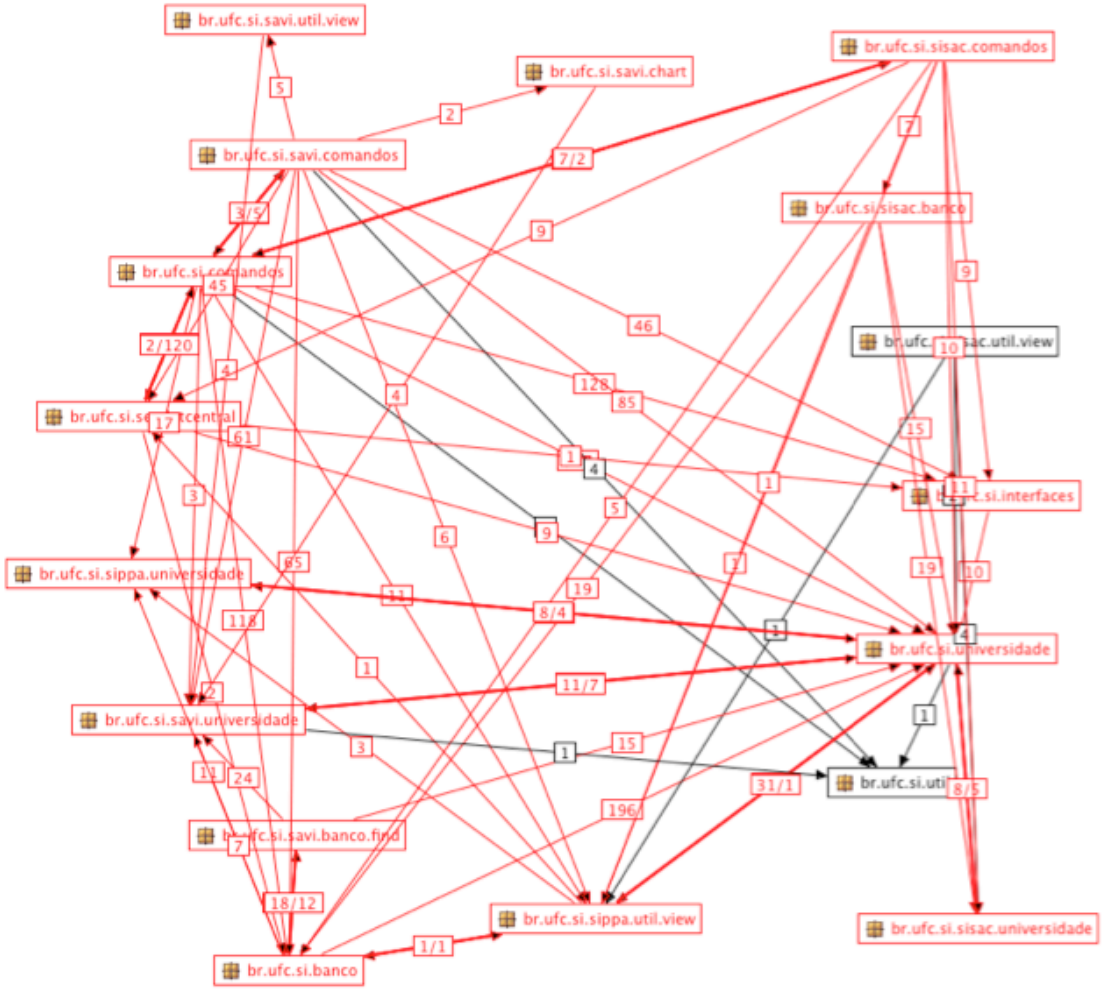

Figura 5 - Relacionamento entre pacotes do sistema legado

Nesse ponto podemos observar que além do sistema legado principal, o SIPPA, haviam sido desenvolvidos dentro do mesmo projeto e do mesmo banco de dados, duas outras aplicações: o SAVI (Sistema de Avaliação Institucional) e o SISAC (Sistema de Atividades Complementares). Esses sistemas também fazem parte do conjunto de aplicações utilizado pelos membros do campus e por terem sido desenvolvidos dentro da mesma aplicação acabaram gerando um alto acoplamento entre os sistemas. Isso também pode ser observado na Figura 6.

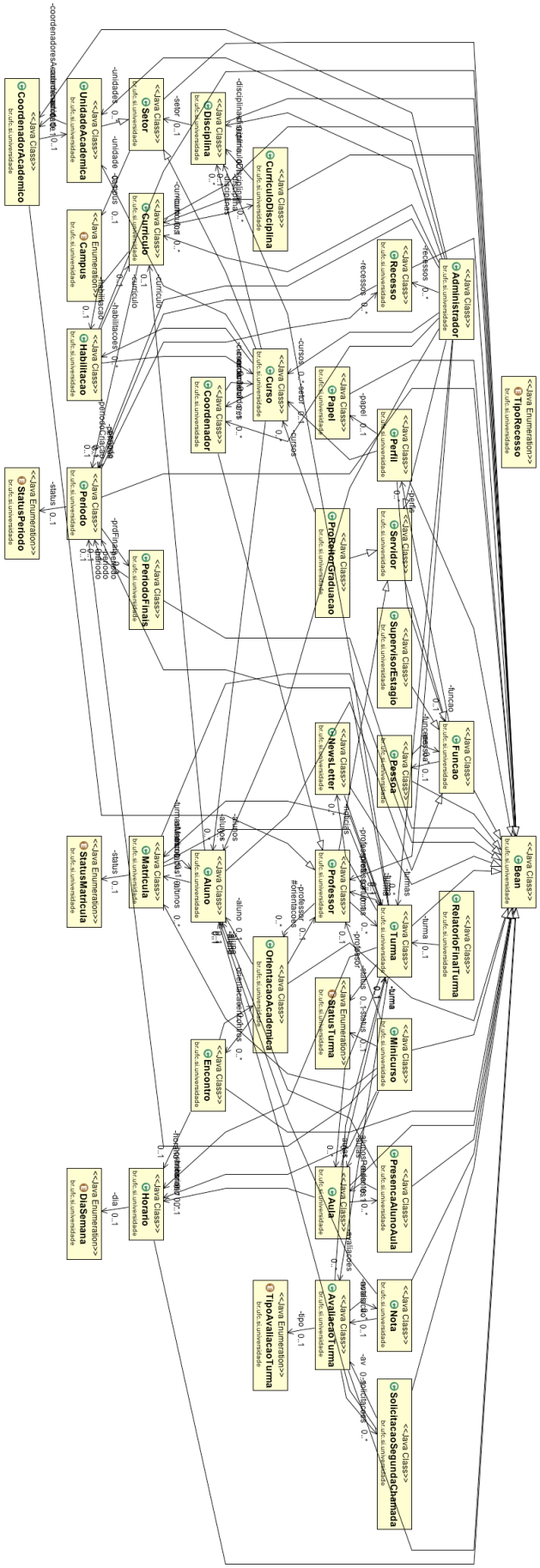

Figura 6 – Relacionamento entre as entidades de negócio do sistema legado

A arquitetura do sistema legado foi orientada segundo o padrão MVC (*Model-view Controller*) e de maneira geral, o projeto está organizado na seguinte estrutura: JSPs, Comandos e ActionFinds. Os JSPs são responsáveis pela parte de visualização da aplicação, o *View* do MVC, os comandos responsáveis pela lógica da aplicação sendo a camada de controle (*Controller* do MVC) intermediando a comunicação entre a *View* e o *Model*, representado aqui pelos ActionsFinders. Os ActionFinders mapeiam as consultas relacionais realizadas pela aplicação em entidades do negócio.

Após essa análise inicial percebeu-se que a camada mais isolada do restante da aplicação era justamente o *Model*. Devido a isso foi constatado que havia uma possibilidade de reuso para essa camada, contudo, é necessário atentar-se que mesmo sendo a camada mais isolada da aplicação, nela ainda era possível verificar *bad smell* como, por exemplo, baixa coesão.

Um dos problemas mais notórios dessa camada era a existência de vários métodos que retornavam uma mesma entidade, mas com atributos específicos preenchidos, ou seja, existiam métodos que retornavam, por exemplo, um aluno, somente com as propriedades do Aluno e outra que retornava as mesmas propriedades do aluno, com as propriedades da entidade Pessoa do Aluno, dificultando assim a utilização dos métodos existentes pois era necessário conhecer o comportamento dos métodos.

Apesar de todos os problemas e dificuldades citados anteriormente, decidiu-se aproveitar essa camada pelo potencial de reuso que ela apresentava. Ela apresentava um alto índice de isolamento se comparada às outras camadas, além de possuir um alto nível de complexidade das consultas SQL e mapeamentos do banco. Também possuía regras de normalização dos índices do banco, algo que normalmente não é de responsabilidade das aplicações.

Outro fator importante foi o fato de não saber se o banco de dados da aplicação possuía *stored procedures* ou *triggers*, o que poderia gerar falhas e inconsistências que teoricamente só seriam descobertas em tempo de execução e preferiu-se então, por esses motivos, reutilizar essa camada isolando-a, através da criação de uma camada que a encapsulava.

6.4 Implementação dos serviços

Agora, com um novo projeto criado para o desenvolvimento da aplicação dos serviços, a camada do *model* do SIPPA foi reutilizado para a nova aplicação e então os seguintes passos foram realizados:

- Refatoração da camada do *model*.

  A camada do modelo que foi reutilizada possuía uma classe chamada *DAOFactory* cujo principal comportamento era ser responsável pelas geração das consultas SQL ao banco e relacioná-las ao seu respectivo ActionFinder para que esse por sua vez pudesse mapear a consulta em uma entidade de negócio.

  A principal ação aqui foi refatorar a classe em questão que possuía quase três mil linhas e muitas responsabilidades em classes menores e de responsabilidades mais explicitas. Utilizando o padrão Extract Method[3], os métodos das classe foram movidos para novas classes que iriam possuir responsabilidades especificas de acordo com o retorno da entidade de negócio do método.

DAOFactory.java at 11/25/13 10:13 AM

Metric	Value
Lines of Code	2,918
Number of Characters	160,531
Number of Comments	213
Number of Constructors	0
Number of Fields	0
Number of Lines	3,561
Number of Methods	242

Figura 7 - Classe DAOFactory antes da migração / refatoração

[3] Extract Method: http://www.refactoring.com/catalog/extractMethod.html

br.ufc.quixada.dao.impl at 11/25/13 10:16 AM

Metric	Value
⊟ Lines of Code	2,974
AlunoDAOImpl.java	281
AulaDAOImpl.java	91
CoordenadorAcademico.java	5
CoordenadorDAOImpl.java	197
CurriculoDAOImpl.java	40
CurriculoDisciplinaDAOImpl.java	79
CursoDAOImpl.java	216
DiarioDeAulaDAOImpl.java	55
DisciplinaDAOImpl.java	111
HabilitacaoDAOImpl.java	115
HorarioDAOImpl.java	61
MatriculaDAOImpl.java	32
NotaDAOImpl.java	34
PessoaDAOImpl.java	49
PlanoDeAulaDAOImpl.java	53
PlanoDeEnsinoDAOImpl.java	112
ProfessorDAOImpl.java	143
ProgramaDisciplinaDAOImpl.java	64
RecessoDAOImpl.java	50
SaviDAO.java	592
SisacDAOImpl.java	53
SolicitacaoSegundaChamadaDAOImpl.java	59
TurmaDAOImpl.java	382
UnidadeAcademicaDAOImpl.java	100
⊞ Number of Characters	154,625
⊞ Number of Comments	169
⊞ Number of Constructors	22
Number of Fields	0
⊞ Number of Lines	3,637

Figura 8 - Classe DAOFactory depois da migração / refatoração

- Criação de interfaces de acesso do *model*.

  Mesmo com divisão da classe anterior, a complexidade da manipulação das consultas e dos mapeamentos, ainda era alta. Então assim, optamos por isolar essa camada do restante da aplicação criando uma interface mais simples baseado em padrões de *Driven Domain Design* (DDD), mais especificamente no padrão *Repository* [4].

- Criação da camada de controle da aplicação

  Após a camada de acesso ao banco está isolada, a camada de controle de aplicação foi desenvolvida. Nela adicionamos, quando necessário, as regras de negócio existentes e as validações. Um adendo importante aqui é que devido a não existência dos requisitos da aplicação, a análise das de algumas funcionalidades deu-se através da análise da implementação dos comandos das classes do sistema legado.

[4] Repository Pattern: http://martinfowler.com/eaaCatalog/repository.html

- Criação das interfaces da camada de serviço.

  Por fim, as interfaces dos serviços foram definidas, baseando-se na ideia de URI única da arquitetura REST e somente sendo migrados os métodos que seriam necessários dos ativos identificados para o desenvolvimento na nova aplicação.

- Definição dos contratos

  Para finalizar, os contratos dos serviços foram definidos utilizando JSON [5] para representação dos dados. Definimos duas estruturas básicas: uma em caso de erro e outra em caso de sucesso.

  Na Figura 9 podemos observar a estrutura que foi definida para caso de sucesso e na Figura 10 em caso de erro. Os atributos da estrutura em caso de sucesso variam de acordo com a entidade requisitada, mas a estrutura em caso de erro é fixa mantendo sempre os mesmos atributos.

  A serialização da mensagem é realizada com o auxilio da biblioteca GSON [6] que auxilia a serialização e desserialização de mensagens no formato JSON na linguagem Java e do vRaptor que utiliza a biblioteca em questão. A Figura 11 exibe um trecho do código do serviço responsável por isso.

[5] JSON (JavaScript Object Notation) : http://json.org/
[6] Gson: https://code.google.com/p/google-gson/

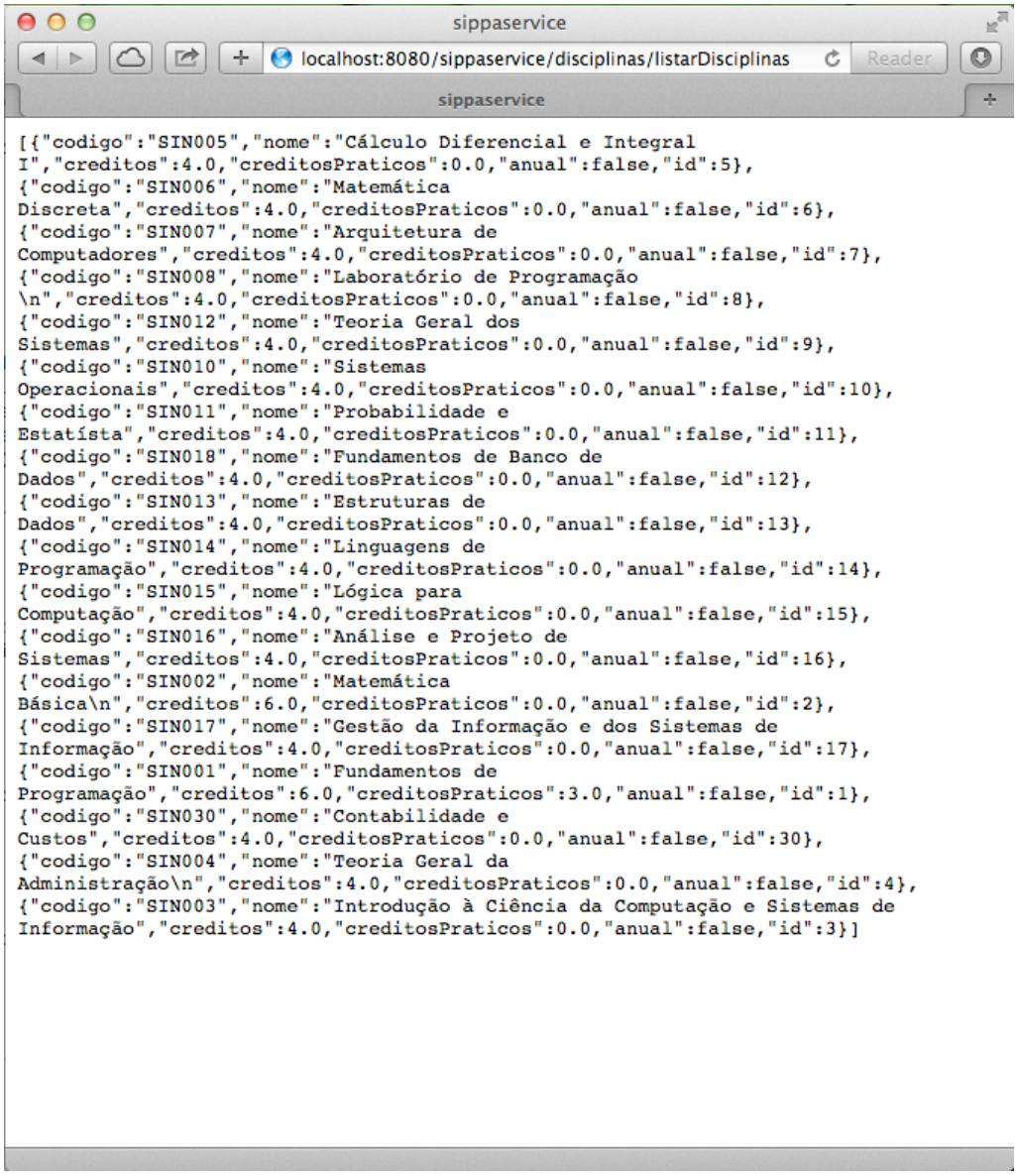

Figura 9 - Contrato do serviço em caso de sucesso

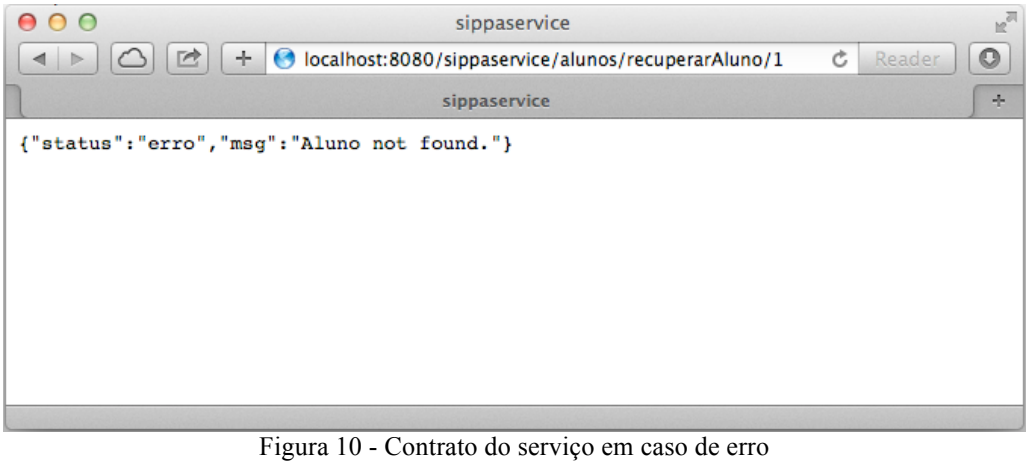

Figura 10 - Contrato do serviço em caso de erro

```
34
35    @Path("/disciplinas/listarDisciplinas")
36    public List<Disciplina> listarDisciplinas() {
37        List<Disciplina> disciplinas = null;
38        try {
39            disciplinas = disciplinaService.listarDisciplinas();
40            result.use(Results.json()).withoutRoot().from(disciplinas).serialize();
41        } catch (BusinessLogicException | DAOException e) {
42            Map<String, String> response = new HashMap<String, String>();
43            response.put("msg", e.getMessage());
44            response.put("status", "erro");
45            result.use(Results.json()).withoutRoot().from(response).serialize();
46        }
47        return disciplinas;
48    }
49
```

Figura 11 - Serialização da mensagem no serviço

Com os serviços necessários criados, o desenvolvimento da aplicação de Pré-Demanda foi iniciado implementando todas as funcionalidades anteriormente definidas.

A aplicação também foi desenvolvida em Java, com a utilização do vRaptor e para a comunicação com o serviço, a biblioteca GSON foi utilizada para fazer a desserialização das mensagens.

Na Figura 12 podemos visualizar o trecho do código da aplicação de Pré-Demanda que é responsável pela desserialização da mensagem da Figura 9.

```
31    public List<Disciplina> listarDisciplinas() throws ConnectionException{
32        URLRequestUtil requestUtil = new URLRequestUtil();
33        requestUtil.setURL(URL_SERVICE + "/listarDisciplinas")
34                .setTypeRequest(TypeRequest.POST);
35
36        String jsonResponse = requestUtil.execute();
37
38        Gson gson = new Gson();
39        Type colletionType = new TypeToken<Disciplina[]>(){}.getType();
40
41        Disciplina[] disciplinas = gson.fromJson(jsonResponse, colletionType);
42        List<Disciplina> lista = new ArrayList<Disciplina>();
43        for (Disciplina disciplina : disciplinas) {
44            lista.add(disciplina);
45        }
46        return lista;
47    }
```

Figura 12 - Desserialização da mensagem no cliente

Como foi forma de facilitar o entendimento de como foi realizada a comunicação e o desenvolvimento da aplicação, o código foi disponibilizado na seguinte URL: https://github.com/bcfurtado/predemanda

6.5 Dificuldades e lições aprendidas

Projetar uma estratégia de migração para um sistema que não se conhece pode ser uma tarefa arriscada. O processo utilizado nesse trabalho foi baseado em uma versão anteriormente elaborada na qual previa um passo extra de integração do sistema legado com os serviços criados, de tal forma que os serviços criados também seriam utilizados no sistema legado para evitar a duplicação de código, prevenindo assim futuras inconsistências.

Contudo verificou-se que essa abordagem não seria adequada devido ao alto acoplamento existente no sistema legado fazendo que um grande esforço fosse demandando para a realização da integração dos novos serviços com o sistema legado. A Figura 13 mostra o primeiro processo de migração criado e a Figura 14 mostra como ficariam os sistemas após a migração, caso esse processo em questão tivesse sido utilizado.

É importante ressaltar também que o processo de migração em questão era de alto risco já que devido a alta complexidade do sistema, o alto acoplamento e a não existências de testes, não seria possível verificar os impactos que as mudanças causariam, podendo causar futuros problemas no sistema legado e consequentemente para a organização.

Com isso apesar de haver replicação de código nas duas aplicações e ser necessário a manutenção das funcionalidades em ambas, isso pode ser compensado através das vantagens que a utilização da nova arquitetura oferece. Dentre elas podemos citar:

- Possibilidade de utilização de novas tecnologias independentes na camada de serviços e nas camadas superiores que as utilizam.
- Geração de serviços reutilizáveis que permitem centralizar e definir comportamentos e regras de negócio para todas as aplicações que utilizam os serviços.
- Esforço relativamente equivalente de execução do processo de migração ao de inserção de funcionalidades no sistema legado.

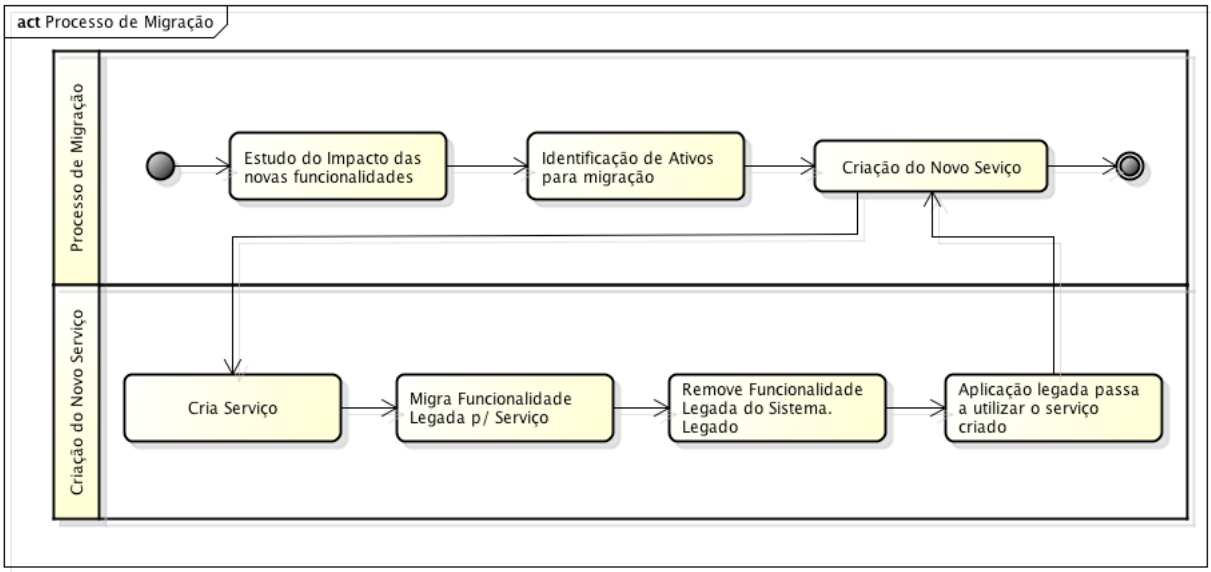

Figura 13 - Primeiro processo de migração

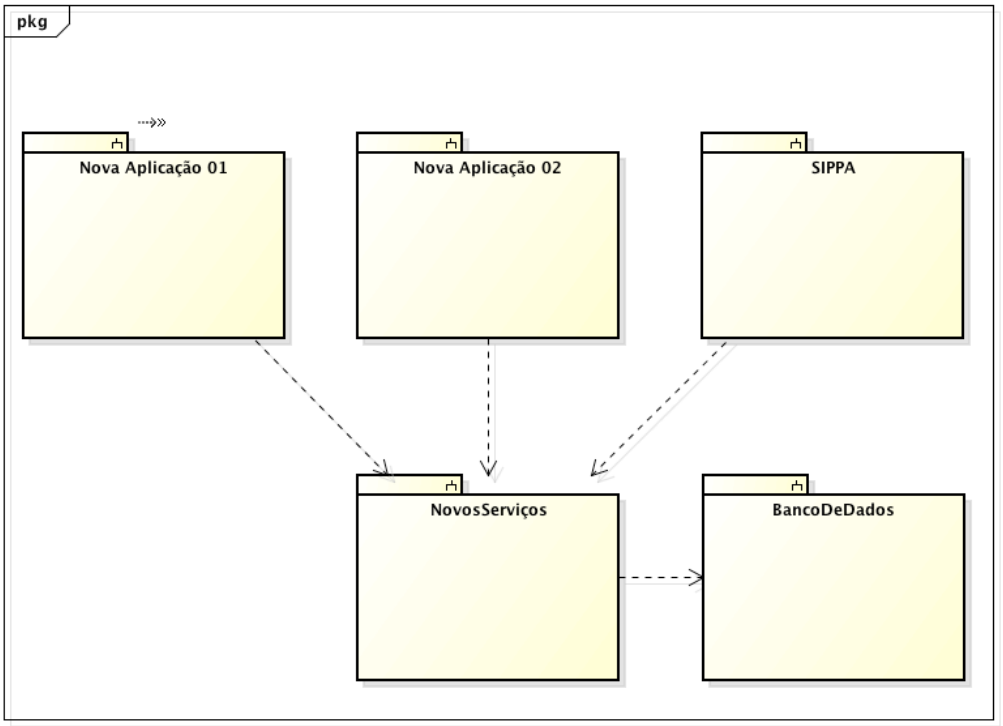

Figura 14 - Sistemas após a migração utilizando o primeiro processo de migração

A clara definição e disposição dos serviços nas URI's também contribuiu para o sucesso da integração, pois permitia a identificação explicita dos recursos/ativos fazendo com que os serviços se tornassem facilmente descobertos e seu comportamento, facilmente verificável.

Outra dificuldade encontrada na execução do processo foi o fato do código do sistema legado disponibilizado para realização da migração estar fragmentado, ou seja, não foi possível executá-lo em um ambiente controlado, fazendo com que não fosse possível a verificação em tempo de execução do fluxo da aplicação e que aliado a não existência de documentação dificultasse o entendimento do sistema legado. Como consequência, a maior parte do tempo da execução do processo foi utilizado para compreender o sistema em questão.

É importante destacar também as imposições ou limitações que uma determinada arquitetura pode oferecer a um sistema. A arquitetura REST, por exemplo, por ser baseada no protocolo HTTP não permite chamada assíncronas, isso por que o próprio protocolo HTTP é síncrono (apesar ser possível através de utilização técnicas). Esse tipo de restrição faz com que os sistemas que utilizarem essa arquitetura ou os serviço expostos por ela, tenham se preocupar, por exemplo, com a latência das requisições.

A arquitetura também se preocupa em fazer com os serviços possam ser descobertos facilmente. Esse que é um dos princípios de SOA, em nosso caso foi solucionado através da indexação dos serviços em uma URI próprio serviço, como pode ser visto na Figura 15.

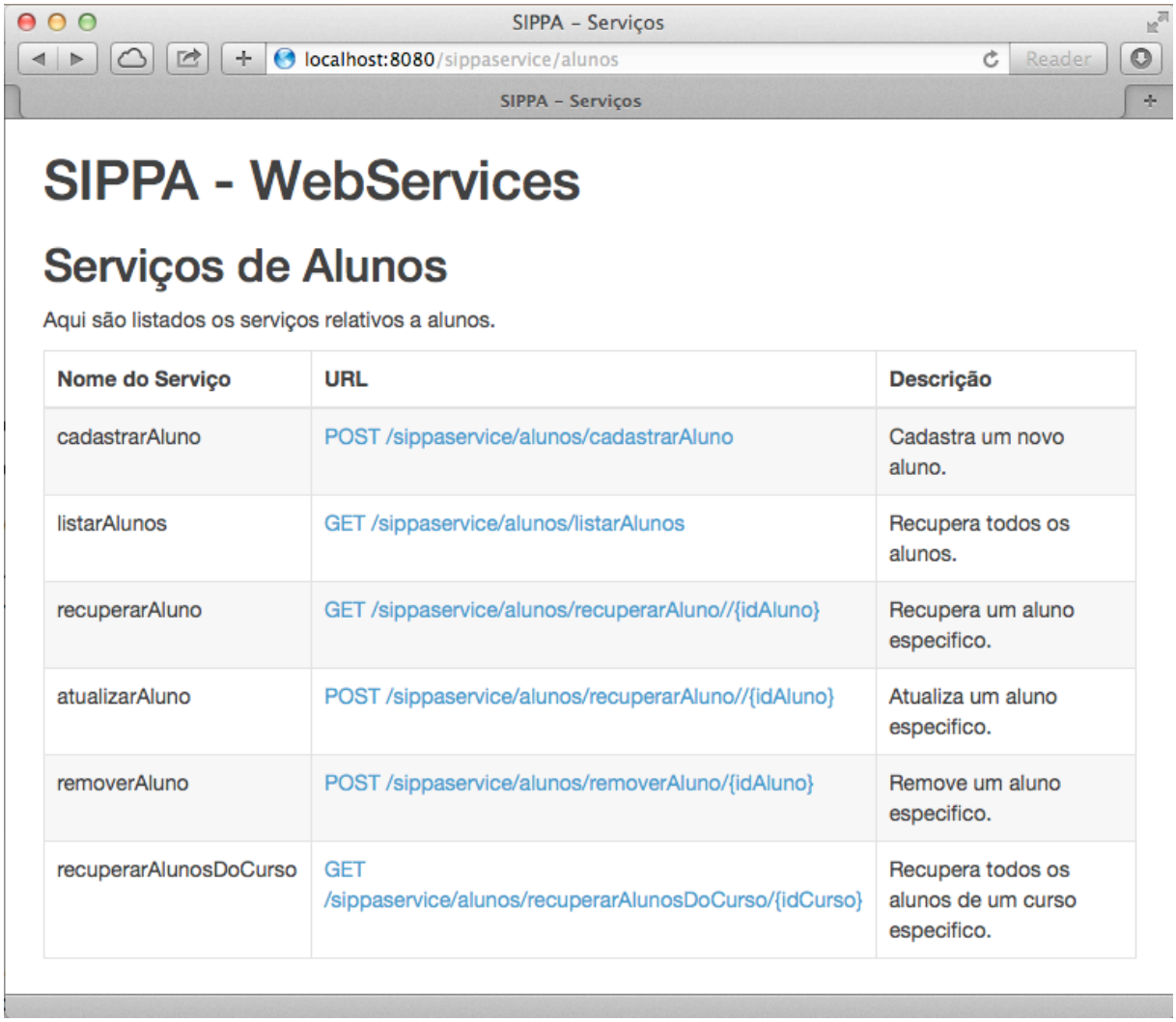

Figura 15 - Página de descoberta dos serviços

7 AVALIAÇÃO DOS SERVIÇOS DESENVOLVIDOS

Os serviços criados nesse trabalho foram projetados com o foco em serem reusáveis pelo fato disso ser um dos princípios de SOA, mas não somente por isso. Criar serviços reusáveis implica em diminuir o trabalho e o esforço dos desenvolvedores, consequentemente diminui o tempo de desenvolvimento, facilitando assim o seu trabalho. Também tem consequência direta no aumento da qualidade do produto final, já que em tese algo que foi utilizados várias vezes já foi testado e o seu comportamento assegurado. Por isso, nessa seção iremos avaliar os serviços criados, criando cenários nos quais esses serviços possam ser reusados em outras aplicações.

Antes de iniciarmos os possíveis cenários de reutilização é necessário relembrar quais serviços foram criados e que poderão ser reusados. Conforme foi exposto na seção 6.1, os seguintes serviços das seguinte entidades foram migrados: Aluno, Coordenador, Curso, Disciplina e Pessoa.

Com esses serviços em mente, uma aplicação que poderia ser desenvolvida, seria uma aplicação móvel focada para alunos no qual teria como funcionalidade principal a consulta a informações das turmas das disciplinas nas quais os alunos estão matriculados auxiliando os alunos a se manterem frequentemente atualizados sobre informações como frequência, notícias e notas, por exemplo, que são enviadas pelos professores para as suas turmas.

Para o desenvolvimento dessa aplicação as seguintes entidades de negócio, em forma de serviço, precisam ser disponibilizadas: Aluno, Turma, Disciplina. Caso o desenvolvimento dessa nova aplicação fosse realizado, então o processo de migração descrito nesse trabalho iria ser disparado.

Disparado o processo, iria ser verificado, na fase de Identificação de Ativos, quais seriam as entidades de negócio que já foram e quais teriam que ser migrados. Em nosso cenário, somente os serviços relacionados à entidade Turma teriam de ser migrados, nos poupando trabalho através do reuso dos serviços já criados.

É importante destacar que nem todos os serviços de uma determinada entidade podem ter sido migradas, já o processo prevê que somente as funcionalidades que forem ser utilizadas é que devem ser migradas. Um exemplo disso pode ser visto na aplicação de Pré-Demanda, na qual a maioria das funcionalidades migradas são relativas somente à leitura de

dados. Funcionalidades relativas à remoção, atualização e criação não foram migradas devido não serem necessárias para o desenvolvimento do sistema.

Outra aplicação que pode ser desenvolvida, agora focando no Professor, é a uma aplicação móvel para a realização das frequências das turmas. Os serviços necessários quando se considera, por exemplo, que a aplicação anterior já foi desenvolvida, temos somente duas entidade que devem ser migrada para desenvolvimento: Aula e Professor.

É possível verificar que os serviços migrados na primeira interação, em tese, seriam reusados em todas as funcionalidades aqui citadas, o que demostra uma grande capacidade de reuso dos serviços criados.

Também verifica-se que inicialmente existe um maior esforço para migração de um conjunto de funcionalidade, mas posteriormente a cada nova interação do processo, o esforço tende a diminuir já algumas funcionalidades já foram migradas, fazendo o desenvolvimento de novos sistemas, possa ser mais desacoplado e consequentemente mais fácil e rápido já que não dependerá mais do sistema legado.

8 CONSIDERAÇÕES FINAIS

Este trabalho apresentou a migração de um sistema legado para SOA procurando demostrar as vantagens que a utilização dessa arquitetura pode oferecer, especificamente a facilitar o desenvolvimento de aplicações de novas aplicações e de processos de negócio já existentes.

Dessa forma, expomos durante o desenvolvimento desse trabalho as dificuldades encontradas na migração de sistemas e as decisões que foram tomadas como forma de adequar a ideia que SOA prega em um contexto no particular da aplicação legada.

Vimos que a utilização dessa abordagem aumenta a capacidade de reuso, consequentemente diminuindo o tempo de desenvolvimento e aumentando a qualidade das aplicações que utilizam os serviços. Além disso, essa abordagem também aumenta e agrega mais valor ao software da organização, trazendo também maior longevidade para o sistema legado e diminuição dos custos de manutenção dos novos sistemas.

Um exemplo disso pode ser visualizado na aplicação de Pré-Demanda, que se tivesse sido desenvolvida como uma funcionalidade extra do sistema legado, teria aumentado a complexidade do sistema legado além já iniciar o seu ciclo de vida com um grande custo para manutenção e evolução.

É importante destacar que o processo de migração e evolução descrito nesse trabalho é um algo constante dentro do ciclo de vida do sistema legado e que ele pode demorar anos até que seja totalmente migrado. O importante é sempre é termos em mente uma estratégia que nós permita evoluir o sistema e assim aumentar o valor que o software agrega a organização, se não, o software estará fadado a erosão.

Sistemas realmente grandes e verdadeiramente complexos sempre terão grandes desafios de desenvolvimento, design, performance, manutenção, escalabilidade, etc. Todas essas dificuldades, tornam o desenvolvimento de software algo complexo e cada vez mais desafiador.

Por fim, concluímos que não existem formas especificas para o sucesso do desenvolvimento e migração de sistemas, o que torna o processo de desenvolvimento de sistemas, um processo criativo, apesar do fato que a computação, por si só, ser uma ciência exata.

O próximo passo para esse trabalho é a possibilidade da realização de um estudo mais completo das consequências diretas e indiretas que essa arquitetura e o seu processo implicam nas arquiteturas das aplicações subsequentes.

É possível ainda a criação de um framework próprio ou projeto em branco pré-configurado para o desenvolvimento de novas aplicações baseadas na arquitetura atual, o que facilitaria ainda mais a manutenção das novas aplicações, por possuírem base de código similar.

Mais estudos também são necessários para verificação da necessidade de migração ou melhoramentos na camada de banco de dados da aplicação, pois este também é um componente importante no sistema.

Por fim, também existe a oportunidade de continuar a migração do sistema legado para que se possa refinar ainda mais o processo.

REFERÊNCIAS

ALMONAIES, A; ALAFI, M; CORDY, J; DEAN,R.. **Towards a framework for migrating web applications to web services.** In Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research (CASCON '11). Riverton, NJ, USA: 2011.

ARCELLI, F.; TOSI, C.; ZANONI, M.. **Can design pattern detection be useful for legacy system migration towards SOA?** In Proceedings of the 2nd international workshop on Systems development in SOA environments - SDSOA '08. New York, New York, USA: ACM Press, 2008.

BISBAL, J.; LAWLESS, D.; GRIMSON, J.. **Legacy information systems: issues and directions**. IEEE Software, v. 16, n. 5, p. 103-111, 1999.

CALLE, G.; MARTÍNEZ, E.A.; TZAGARAKIS, M.; KARACAPILIDIS, N.. **The Dicode Workbench: A Flexible Framework for the Integration of Information and Web Services**. In Proceedings of the 14th International Conference on Information Integration and Web-based Applications & Services (IIWAS '12). ACM, New York, NY, USA, p. 16-25.

DAIGNEAU, R.; **Service Desing Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services.** 1 ed. Addison-Wesley Professional, 2011.

ERL, T.. **SOA Princípios de design de serviços.** São Paulo: Prentice Hall, 2009.

GOULART, A. M. C.. **O conceito de ativos na contabilidade: um fundamento a ser explorado.** Revista contabilidade financeira, São Paulo, v. 13, n. 28, Abril 2002. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1519-70772002000100004&lng=en&nrm=iso>. Acesso em 05 de jul. 2013.

HOHPE, G.; WOOLF, B.. **Enterprise integration patterns: Designing, building, and deploying messaging solutions**. 1 ed. Addison-Wesley Professional, 2004.

JOSUTTIS, N.. **SOA na Prática**. Rio de Janeiro : O'Reilly Media, 2008.

KONTOGIANNIS, K.; LEWIS, G. A.; SMITH, D. B.. **A research agenda for service-oriented architecture.** In Proceedings of the 2nd international workshop on Systems development in SOA environments - SDSOA '08. New York, New York, USA: ACM Press, 2008.

LEWIS, G. et al. **Service-Oriented Migration and Reuse Technique (SMART)**. In 13th IEEE International Workshop on Software Technology and Engineering Practice (STEP'05). Budapest: IEEE, 2005.

NEWCOMER, E.; LOMOW, G.. **Understanding SOA with Web Services (Independent Technology Guides)**. Addison-Wesley Professional, 2004.

PAUTASSO, C.; ZIMMERMANN, O.; LEYMANN, F.; **Restful Web Services vs. "Big"' web services**. Proceeding of the 17th international conference on World Wide Web - WWW '08. New York, USA: ACM Press, 2008.

RAZAVIAN, M.; LAGO, P.. **A survey of SOA migration in industry**. In Proceedings of the 9th international conference on Service-Oriented Computing (ICSOC'11). Berlin, Heidelberg: Springer-Verlag, 2011. P. 618-626.

RAZAVIAN, M.; LAGO, P.. **Towards a Conceptual Framework for Legacy to SOA Migration.** In: Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops. Stockholm, Sweden: Springer Berlin Heidelberg, 2010. p. 445-455.