Universidade Federal de Pernambuco
Centro de Informática

Uma proposta de arquitetura para serviço de Chat em Rede Social Educacional

Trabalho de Graduação

Aluno: Guilherme Oliveira Cavalcanti

Orientador: Fernando José Castor de Lima Filho

Recife, 13 de Dezembro de 2011

Universidade Federal de Pernambuco
Centro de Informática

Guilherme Oliveira Cavalcanti

Uma proposta de arquitetura para serviço de Chat em Rede Social Educacional

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Fernando José Castor de Lima Filho

Índice

Figuras

Tabelas

1 Introdução

O sincronismo faz parte da natureza da World Wide Web. O usuário interage com a interface Web através do navegador. Este último realiza uma requisição para o servidor o qual responde com uma representação do estado do sistema naquele momento. Fundamentalmente um processo síncrono. Porém, para alguns tipos de aplicação, tal representação do estado não reflete a natureza dinâmica do sistema. Mesmo com o uso de técnicas como AJAX (Asynchronous JavaScript and XML) (GARRET, 2005), a natureza do ciclo requisição-resposta não muda. Ou seja, há sempre a necessidade da requisição para se obter uma nova representação do estado do sistema.

A Web assíncrona é fundamentalmente diferente. Neste cenário o servidor é capaz de enviar novas representações de estados do sistema assim que as mesmas são detectadas. Interações assíncronas são importantes em aplicações colaborativas, onde o estado do sistema é alterado com frequência por clientes diferentes. Nesses tipos de aplicações tornar-se-iam inviáveis, ou extremamente ineficientes, aplicações com requisitos de tempo real.

A Rede Social Educacional Redu (MELO, 2010) é uma plataforma de ensino a distância que faz uso de funcionalidades de tempo real como forma de diminuir a distância transacional. O conceito de distância transacional denota o diálogo entre professores e aprendizes que estão espacialmente separados (MOORE, 2007). Para engajar aprendizes num diálogo síncrono faz-se uso da funcionalidade de Chat par a par. Aumentando este tipo de diálogo é possível diminuir a distância transacional entre professor e aprendiz fazendo com que o segundo necessite de menos autonomia para se engajar no processo de aprendizagem. Todos estes fatores em consonância diminuem a barreira de entrada para novos usuários em ambientes virtuais de aprendizagem. Além disso contribuem para a diminuição das taxas de evasão em ambientes virtuais de aprendizagem.

1.1 Problemas

Até o momento da criação deste trabalho a funcionalidade de Chat do Redu é obtida através do Pusher (PUSHER, 2011). O Pusher é um serviço Web, disponibilizado por meio de um *webservice*, que se propõe a facilitar a criação de funcionalidades em tempo real através do uso de uma API REST (FIELDING, 2000) e de WebSockets ("The WebSocket API", 2011). As decisões de projeto para esta solução se apresentaram ideais para os estágios iniciais do produto. Porém, com a evolução do mesmo, tópicos como performance e custos vieram a tona. Logo, fez-se necessário repensar tais decisões para que exista a evolução do produto.

Dadas as decisões de projeto tomadas para a criação da arquitetura atual, foram identificados dois problemas-chave que se mostraram empecilhos para o crescimento do produto Redu. Estes problemas são o objeto de estudo do presente trabalho e serão tomados como premissas para as decisões de projeto da nova arquitetura.

1.1.1 Problemas de performance

Devido a limitações da API *client-side* do Pusher, é necessário realizar uma requisição HTTP por autenticação. Este procedimento é descrido em detalhes na Seção 3.2.1 mas, em linhas gerais, são necessárias $2n + 2$ requisições HTTP no primeiro carregamento, onde n é igual ao número de contatos da rede estendida do usuário. Logo, num caso de o usuário possuir 20 contatos serão necessárias 42 requisições para o carregamento completo do Chat. Para se ter uma ideia relativa da magnitude deste número são necessárias 55 requisições HTTP para arquivos estáticos (imagens, JavaScripts e CSS), no carregamento da página inicial da aplicação.

Uma vez que o Chat é carregado e inicializado, o usuário não percebe a lentidão de imediato. Porém, caso o seja feita uma nova requisição (por exemplo, mudança de página) antes que a requisições de autenticação sejam completadas tal lentidão é percebida. Isto acontece pois os *browsers* modernos permitem, em média, 6 conexões concorrentes abertas para um mesmo host ("Browserscope", [S.d.]) criando uma fila para as requisições posteriores. Portanto, caso existam mais de 6 requisições relativas a autenticação, todas as requisições subsequentes serão enfileiradas e executadas apenas quando as requisições de autenticação são respondidas.

Este problema foi identificado através de uma análise dos arquivos de log da aplicação em produção (ver Tabela 1-1). Foi identificado que cerca de 63% das

requisições feitas num período de 32 dias foram direcionadas para o controlador responsável pela autenticação no Chat (ChatController#auth na Tabela 1-1).

Usuários do sistema com um número significativo de contatos também relataram uma experiência pobre no uso da plataforma por causa da lentidão. Três casos foram analisados e detectou-se que a lentidão foi causada pelo gargalo criado com o número excessivo de requisições de autenticação.

Controlador	Hits	% do Total
ChatController#auth	156145	63.7%
EnvironmentsController#preview	32221	13.2%
UsersController#home	5834	2.4%
BaseController#site_index	5282	2.4%
CoursesController#show	3301	1.3%

Tabela 1-1: Requisições por controlador

1.1.2 Custo

Como um dos componentes do sistema é fornecido por meio de um *webservice* de terceiros (Pusher), existe um custo associado ao mesmo. O custo do Pusher varia de acordo com o número de mensagens e o número de conexões simultâneas (usuários on-line no Chat). Com a evolução do produto Redu, foi identificado que estes custos iriam impactar de forma demasiada o preço para o usuário final, aumentando as barreiras de entrada para os mesmos.

Na Figura 1-1 é mostrada a relação entre usuários simultâneos e o custo gerado pela funcionalidade de Chat. O número de conexões concorrentes, embora não seja a única variável envolvida na definição do custo, é a que mais contribui para o seu crescimento. Também é importante notar que existe um limiar entre 500 e 5000 usuários simultâneos que irá saturar o servidor da aplicação devido ao alto número de requisições de autenticação.

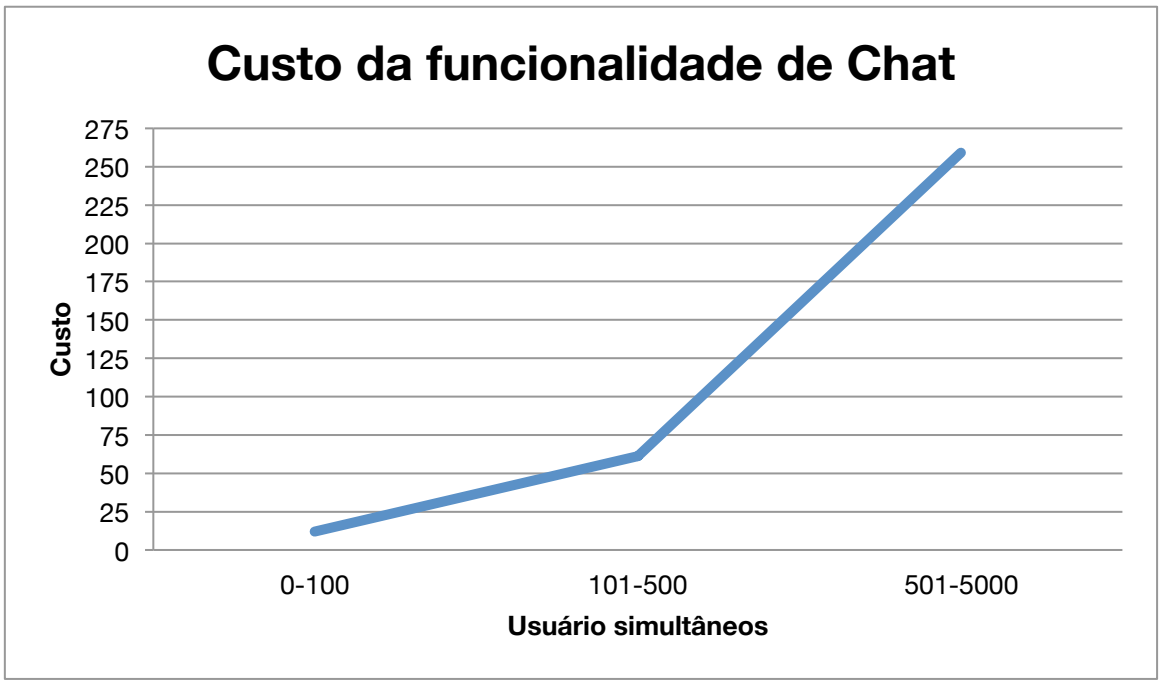

Figura 1-1: Custo por usuários simultâneos

1.1.3 Experiência do usuário inadequada no uso off-line

Cenários futuros irão requerer que o Redu funcione em modo off-line dentro de LANs em comunidades remotas (ver Figura 1-2). Por se tratar de um *webservice* de terceiros, o Pusher não poderia operar sem acesso constante à Internet. Dentro deste cenário uma aplicação replicada irá operar no servidor da LAN remota e periodicamente realizará a sincronização dos dados com a aplicação principal (fora da LAN).

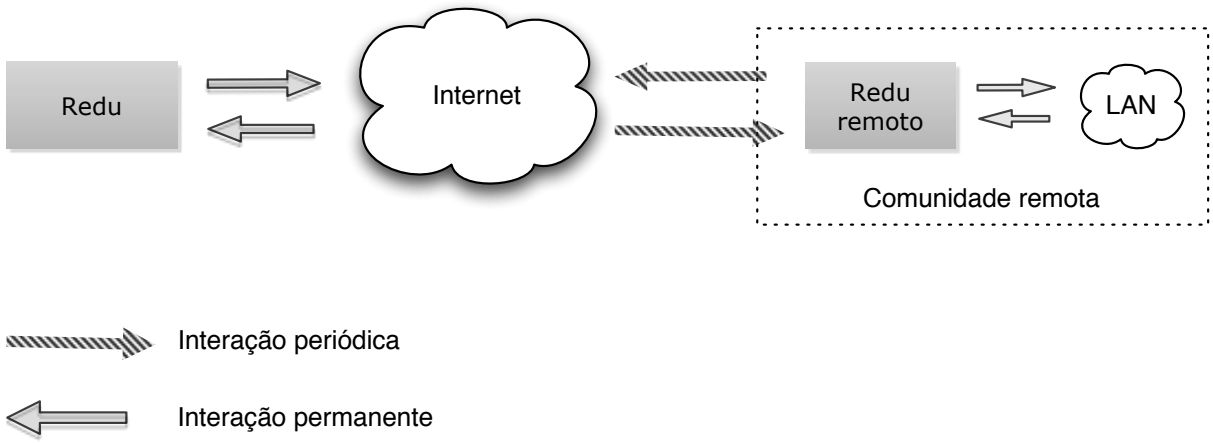

Figura 1-2: Redu remoto

Uma vez que a funcionalidade de Chat gera alto valor para os usuários da Rede Social Educacional, torna-se crítico que a mesma seja oferecida mesmo sob

tais condições. Neste cenário os usuários das comunidades remotas poderão conversar através do Chat entre si e terão contato com os demais usuários do Redu nos momentos que a conexão com à Internet existir.

1.2 Objetivos

O objetivo deste trabalho envolve o projeto, documentação e prova de conceito de uma nova arquitetura que contorne problemas relacionados a custo e desempenho da funcionalidade de Chat no estágio atual do produto. As decisões de projeto da nova arquitetura serão justificadas através de critérios bem definidos, documentadas e compartilhadas com os *stakeholders* do projeto.

Com isso pretende-se melhorar um produto real através da diminuição da distância transacional entre professores e aprendizes. A redução da distância transacional foi mostrada como um fator crítico para o bom aproveitamento de sistemas de educação a distância (SHIN, 2004). Também espera-se como resultado o aumento da sinergia entre academia e os arranjos produtivos locais através da transferência de expertises em ambos os sentidos.

1.3 Abordagem proposta e trabalhos relacionados

Na maior parte dos processos de desenvolvimento, decisões de projetos não são documentadas de maneira explícita. Tais decisões estão implícitas no modelo que o arquiteto cria (TYREE, 2005). Para que tais decisões sejam entendidas pelos *stakeholders* do projeto, está dentro do escopo deste trabalho documenta-las.

O modelo utilizado para definir e documentar tais decisões é proposto em (TYREE; AKERMAN, 2005). Tal modelo é uma variação de modelos tais como REMAP (*Representation and Maintenance of Process Knowledge*) e DRL (*Decision Representation Language*) .

Levando em consideração os problemas apontados na Seção 1.1, três objetivos foram traçados. Para cada objetivo descrito, possíveis soluções serão propostas e confrontadas com critérios bem definidos para eleger as melhores decisões de projeto da nova arquitetura. Cada decisão será relacionada com suas implicações, argumentos e limitações e serão compartilhadas com os *stakeholders* do projeto.

1.4 Estrutura do documento

Este trabalho está dividido em seis capítulos organizados como segue:

No Capítulo 2 serão apresentadas as fundamentações teóricas que justificam o uso de interações síncronas no contexto de educação a distância. Também serão mostradas as implicações desse tipo de funcionalidade na taxa de evasão de cursos a distância.

No Capítulo 3 será apresentada a Rede Social Educacional Redu do ponto de vista de engenharia de software. Os principais componentes do sistema de Chat serão explicados bem como os fatores considerados nas decisões de projeto tomadas.

No Capítulo 4 serão estabelecidas abordagens, critérios e fatores levados em conta nas decisões de projeto para a arquitetura proposta. Tais decisões serão apresentadas de maneira estruturada segundo um modelo conceitual.

No Capítulo 5 as decisões de projeto darão origem à arquitetura proposta. Neste capítulo também serão abordadas as tecnologias utilizadas.

No Capítulo 6 serão apresentados os resultados e subprodutos desenvolvidos no presente trabalho.

2 Fundamentos

Neste Capítulo será apresentado como o uso de atividades síncronas (como Chats, ferramenta de *groupware*, etc.) têm influência em fatores como nível de autonomia e evasão. Também será apresentado o conceito de distância transacional que fundamenta a presença da funcionalidade de Chat no Redu.

2.1 Evasão no ensino a distância

Em 2010, a evasão média em cursos a distância, no Brasil, foi de 18,5% ("Censo ead.br", 2010). Quando focado em cursos de extensão este número aumenta para 29% dos alunos matriculados. Foi constatado que a maior causa para os altos níveis de evasão está ligado ao desconhecimento do aluno em relação a metodologia do ensino a distância.

2.2 Distância transacional

O conceito de transação foi cunhado por Dewey (Dewey and Bentlei, 1949) e conota as interações entre ambiente e indivíduos, bem como os padrões de comportamento apresentados por estes numa determinada situação. Uma vez que a separação temporal e espacial é uma característica intrínseca da educação a distância, existe um espaço psicológico e de comunicação que precisa ser atravessado (Moore, 1997). Tal espaço é chamado de distância transacional. A distância transacional sofre influência de dois componentes: estrutura e diálogo. A estrutura é relacionada com o *design* (projeto) do conteúdo veiculado pela rede social educacional. Já o diálogo, diz respeito às interações construtivas dentre aprendizes e professores.

2.3 Diálogo

Diálogo pode ser entendido como um tipo de interação construtiva entre aprendizes e professores. O caráter construtivo destas interações caracterizam o diálogo e fazem este diferir dos outros tipos de interações. O diálogo, no contexto de educação a distância, deve prover crescimento e construção de conhecimento para ambas as partes.

Nem todas interações entre indivíduos podem ser vistas como um diálogo. Por exemplo, quando o aprendiz, dentro de um curso de educação a distância não tem capacidade de contribuir ou modificar a maneira como o conteúdo é passado diz-se que há pouco ou nenhum diálogo, mesmo que haja interação entre aprendiz e professor.

2.4 Estrutura

A estrutura do conteúdo produzido para educação a distância representa o quão flexível ou rígido o programa é. Assim como o diálogo a estrutura depende do meio através do qual o conteúdo é transmitido (Moore, 1997). Um curso a distância com alto grau de estrutura (estrutura rígida) permite pouca adaptação de acordo com a demanda do aprendiz. Um exemplo de conteúdo altamente estruturado é o de cursos oferecidos através da TV. Uma vez que existe baixo diálogo entre aprendiz e professor, cada palavra e atividade são previamente definidas e o fluxo no qual elas ocorrem não pode ser alterado. Já um exemplo de cursos fracamente estruturado são os oferecidos por meio de ferramentas de vídeo conferência, onde os alunos estão livres para intervir e modificar o fluxo das informações apresentadas.

2.5 Autonomia e distância transacional

A relação entre diálogo e estrutura no contexto de educação a distância define o que é conhecido como distância transacional. A variação na distância transacional, por sua vez, acarreta uma variação na autonomia requerida ao aprendiz para que o mesmo obtenha um bom proveito no processo de aprendizagem.

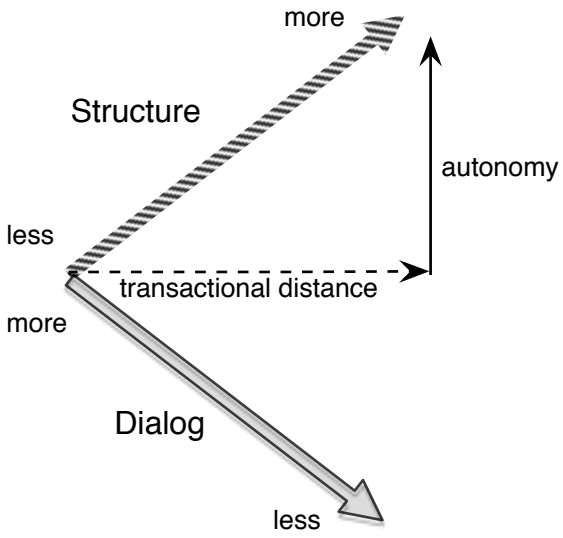

Figura 2-1: Estrutura, diálogo, distância transacional e autonomia

Um programa muito estruturado e com pouco diálogo acarreta um aumento na distância transacional (ver Figura 2-1). Cursos com essas características exigem um maior nível de autonomia ao aprendiz uma vez que o mesmo passa a ser responsável pela regulação do aprendizado. Por outro lado, um programa pouco estruturado e com muito diálogo é mais atraente para aprendizes com pouca autonomia.

3 A Rede Social Educacional Redu

O Redu é uma plataforma de ensino a distância que foi construída sobre uma estrutura de redes sociais. Nela é possível que instituições de ensino e empresas criem seu próprio Ambiente de Ensino e Aprendizagem (AVA), como também permite a criação de cursos para serem oferecidos a distância. Tais funcionalidades são oferecidas como um serviço que pode ser contratado por empresas, instituições e produtores independentes de conteúdo.

É sabido que a maior parte das instituições de ensino, sejam delas publicas, privadas ou corporativas, alegam dificuldades na customização, treinamento e manutenção dos ambientes virtuais de aprendizagem (“Censo ead.br”, 2010). O Redu, até o momento do desenvolvimento deste trabalho, resolve todos estes problemas.

Neste Capítulo serão apresentados os componentes básicos da arquitetura da funcionalidade de Chat do Redu. Além disso serão mostrados os fluxos básicos de autenticação e conversação por meio do Chat. Estes elementos servirão de insumos para as decisões de projeto da nova versão do Chat.

3.1 Arquitetura do funcionalidade de Chat

A comunicação enviada através do Chat é feita por passagem de mensagens segundo o padrão publisher/subscriber. Toda conversa de Chat é um canal ao qual os usuários envolvidos estão inscritos. As mensagens de Chat são primeiramente enviadas para o Redu que, por sua vez, notifica o *webservice* Pusher, que a repassa para os clientes inscritos no canal.

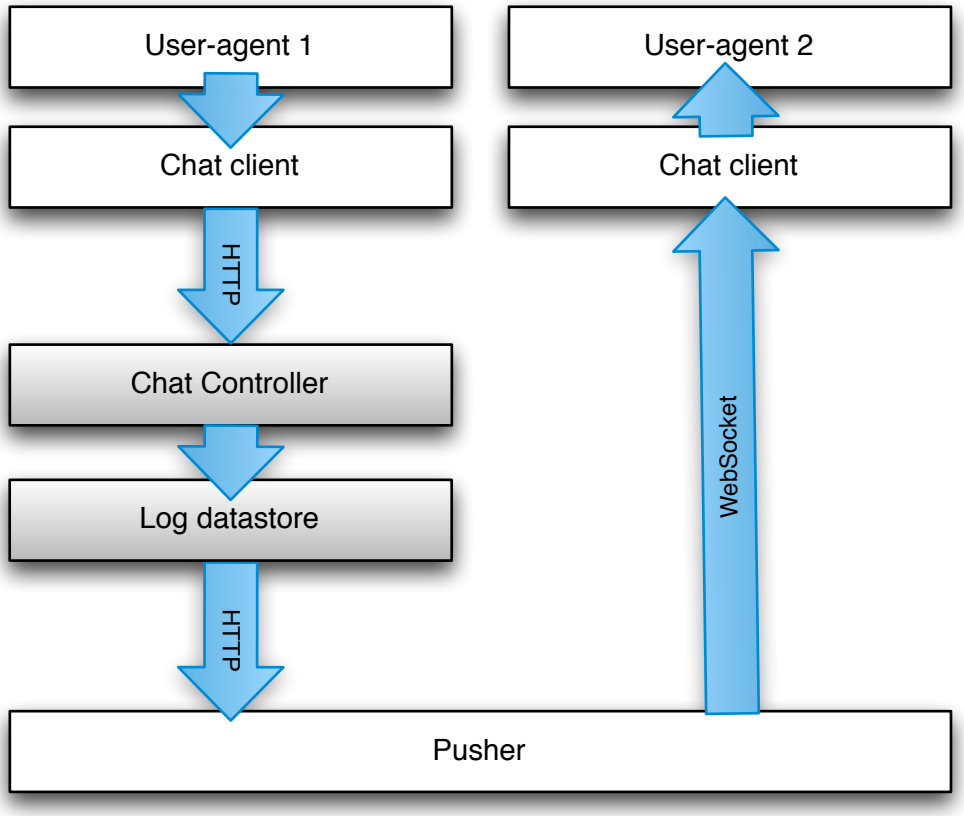

Figura 3-1: Sistema de Chat em tempo de execução (Pusher)

O diagrama da Figura 3-1 representa uma visão dos componentes que fazem parte do Chat em tempo de execução. Os clientes, através de uma interface REST se comunicam com o servidor da aplicação Redu (representado em cinza) que, por sua vez, realiza o log da mensagem e notifica o *webservice* Pusher. O papel do *webservice* Pusher é de notificar os demais clientes conectados a respeito da mensagem enviada. Esta notificação é realizada via WebSocket através de uma conexão *full-duplex* previamente estabelecida.

3.2 Componentes básicos

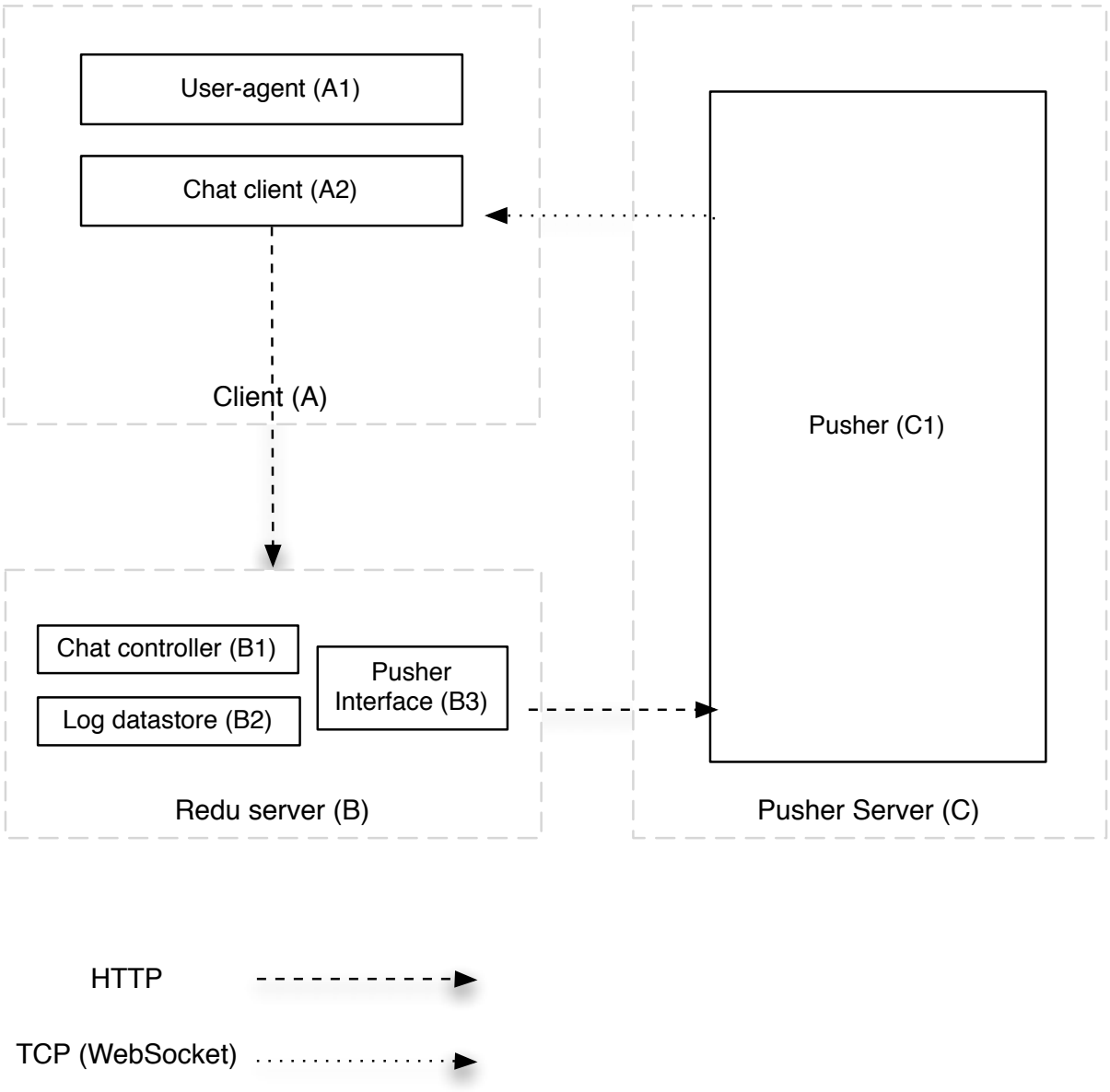

Figura 3-2: componentes do sistema de Chat (Pusher)

O serviço de Chat possui três host básicos (Figura 3-2) o cliente (A), o servidor do Redu (B) e o servidor do Pusher (C). O componente ChatClient (A1) é responsável por autenticar os usuários on-line nos canais de Chat de seus contatos através de uma API JavaScript que gera requisições HTTP para o servidor do Redu (B). Também é responsabilidade deste componente enviar mensagens de Chat quando um usuário as digita.

O servidor do Redu (B), ao receber uma nova mensagem de Chat, realiza o log da mesma (transação de SGBD) e notifica o *webservice* Pusher (C) que uma nova mensagem foi enviada num determinado canal. O componente Pusher, por

sua vez, notifica todos os usuários conectados àquele canal que uma nova mensagem foi enviada.

A comunicação entre cliente (A) e servidor do Redu (B) e entre o servidor do Redu (B) e o *webservice* Pusher (C) é feita através de uma API REST (HTTP). Já a comunicação entre Pusher (C) e o clientes (A) é feita através do protocolo WebSocket ("The WebSocket API", 2011).

3.2.1 O cliente de Chat

O ChatClient (A2) é responsável pela autenticação e passagem de mensagens de Chat de um usuário num determinado canal. Este componente faz requisições HTTP para o servidor do Redu (B) e recebe novas mensagens de Chat e informações de presenças através de uma conexão WebSocket com o servidor Pusher (C).

Este componente é escrito em JavaScript e utiliza uma API escrita na mesma linguagem para criar a conexão WebSocket e definir *callbacks* para cada tipo de mensagem que possa ser veiculada pelos canais. A lista de usuários é exibida através de arquivos HTML e CSS gerado pelo ChatClient em tempo de execução.

3.2.2 O componente Redu

O servidor do Redu (B) possui um componente que é responsável por receber a requisições do cliente de Chat, realizar *logging* e repassar as mensagens para o servidor Pusher. Toda sua comunicação com os dois outros componentes é feita através do protocolo HTTP. As operações de log e autenticação são ambas bloqueantes, ou seja, precisam ser finalizadas antes do fluxo da requisição seguir em frente. O componente ChatController faz parte e é acoplado com a aplicação da rede social educacional. Todas as requisições HTTP, sejam do Chat ou do restante do sistema, são enviadas para o mesmo servidor (B).

Este componente é escrito com a linguagem Ruby utilizando o framework Web Ruby on Rails. As mensagens de Chat são armazenadas em um SGBD (sistema de gerenciamento de banco de dados) relacional instalado na mesma máquina do servidor. O SGBD utilizado é o MySQL e é acessado através de um pool de conexões previamente abertas.

3.2.3 O componente Pusher

O servidor Pusher é um *webservice* que é responsável por notificar todos os clientes conectados a um canal a respeito das mensagens que são enviadas por outros cliente. Para tal, este componente é notificado pelo Redu via HTTP e notifica

demais clientes utilizando WebSocket. O uso de Websocket aqui é justificado pela natureza permanente (tipo *socket full-duplex*) da conexão entre os clientes e o Pusher. Tal fato dispensa o uso de técnicas tais como *long polling*, HTTP *polling* e etc.

3.3 Canais

A comunicação enviada através do Chat é feita por passagem de mensagens segundo o padrão *publisher/subscriber*. Tal abstração tornou mais simples a elaboração da primeira versão do sistema.

Existem dois tipos básicos de canais, os canais privados e os canais de presença. Ambos necessitam de autenticação para que um agente se conecte ao mesmo, porém somente os canais de presença comunicam a todos os clientes conectados quando um novo cliente conecta ou desconecta do canal. Os canais de presença são utilizados para notificar o status on-line e off-line de um usuário para todos da sua lista de contato.

3.4 Autenticação

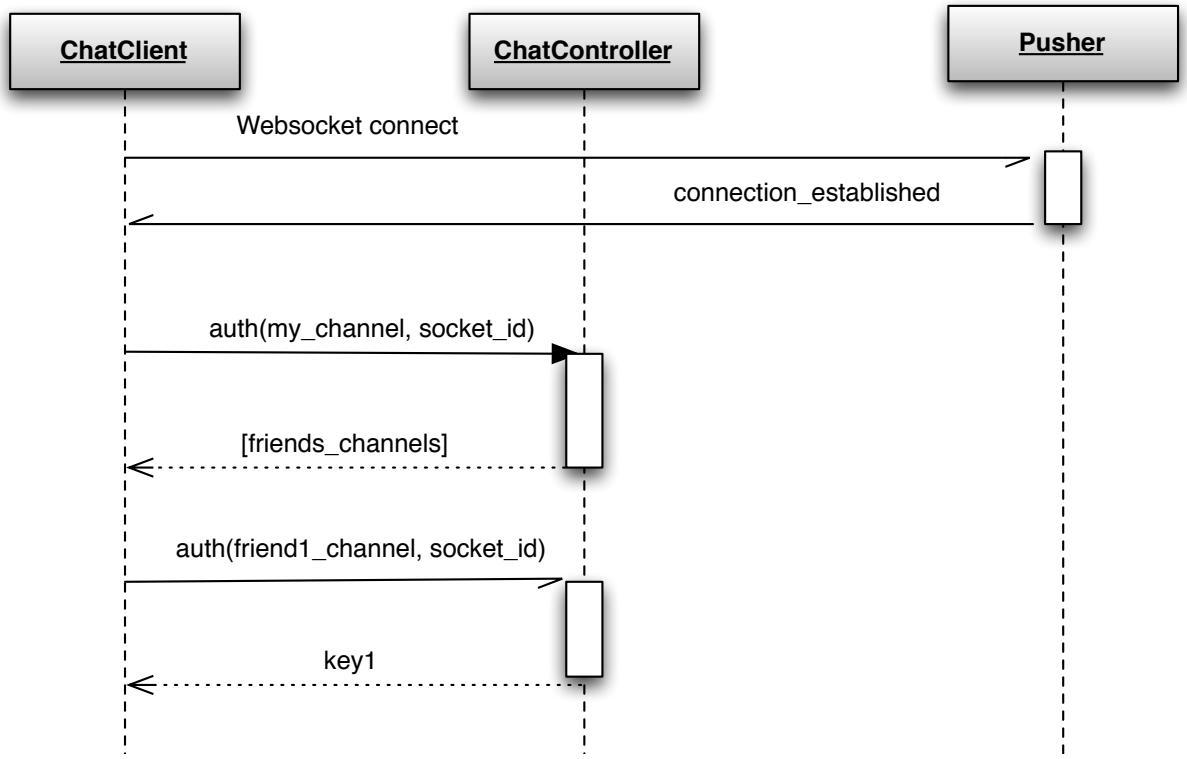

Figura 3-3: diagrama de sequência para autenticação (Pusher)

Antes mesmo da lista de contatos on-line ser exibida, o ChatClient precisa autenticar-se junto ao ChatController. Para isso o fluxo de autenticação é realizado (Figura 3-3). Neste fluxo, as atividades principais são a autenticação no canal do

usuário que está se conectando e a posterior autenticação nos canais de seus contatos. Logo após a renderização da página o ChatClient autentica, via HTTP, junto ao ChatController o usuário em questão. A autenticação é feita através da criação de uma assinatura pública única.

$$signature = HMAC(socket\_id : channel\_id, secret\_key)$$

A chave pública é gerada a partir da criação de um HMAC (*keyed-hash message authentication code*) (TURNER, 2008) utilizando o *SHA256 hexadecimal digest* como algoritmo de encriptação. O resultado de tal função é uma assinatura que poderá ser validada pelo *webservice* Pusher. Para gerar a assinatura são combinados um identificador único para cada agente do usuário (socket_id), o nome do canal ao qual ele está se conectando (channel_id) e uma chave secreta gerada pelo Pusher. Caso a autenticação seja bem sucedida o agente estará apto a receber mensagens pelo canal.

Com a autenticação no seu próprio canal o usuário também receberá a lista de canais aos quais ele deverá se conectar. Estes canais são os canais de seus contatos na Rede Social Educacional. Para cada novo canal o mesmo procedimento de autenticação é realizado. Para cada autenticação uma requisição HTTP é gerada. Este fluxo é realizado uma única vez ao carregar a página inicial do Redu.

3.5 Envio e recebimento de mensagens

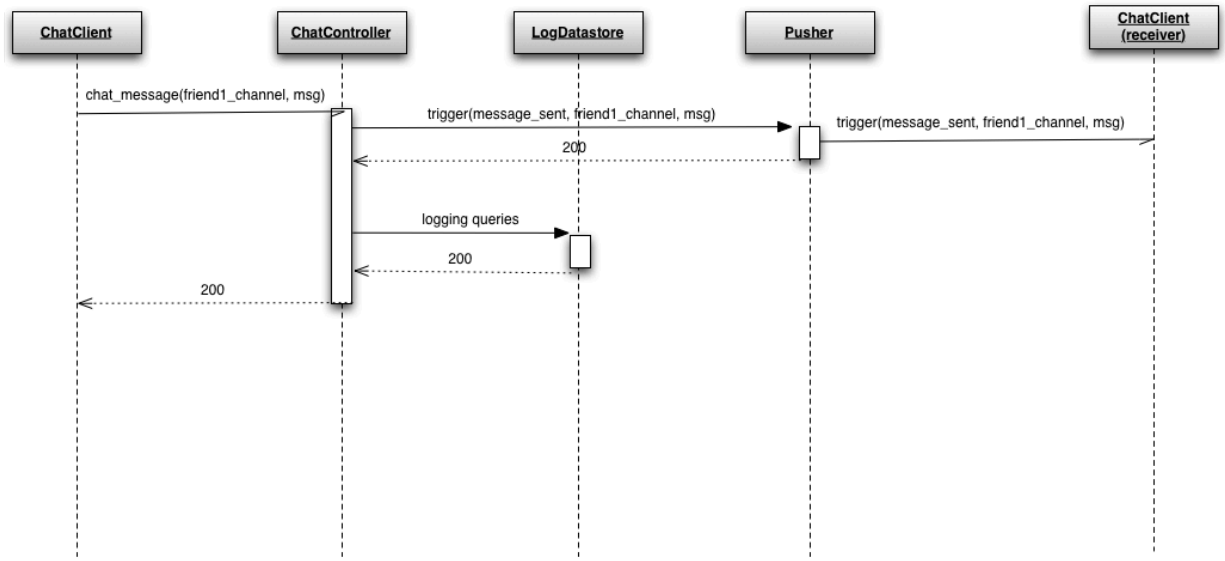

Figura 3-4: diagrama de sequência para envio de mensagens (Pusher)

Supondo que o processo de autenticação já foi realizado, a passagem de mensagens de Chat dentro de um canal se dá através de uma requisição HTTP

simples entre o agente do usuário e o Redu (Figura 3-4). No Redu, a mensagem é persistida no banco de dados e enviada, também através de HTTP para o Pusher. O Pusher, por sua vez, utiliza Websocket para entregar tal mensagem para os usuários conectados no canal em questão. Aqui as operações de log e notificação do Pusher são estritamente sequenciais. Ou seja, a notificação não acontece até o momento em que a mensagem é logada.

4 Decisões de projeto

Baseado nos problemas identificados na Seção 1.1 foram levantadas as seguintes questões que representam as decisões a serem tomadas. Para cada questão existem uma ou mais abordagens é objetivo desta Seção é demonstrar as justificativas por trás da abordagem escolhida.

As decisões estão apresentadas segundo o modelo definido por (TYREE; AKERMAN, 2005). Este modelo propõe um racional a ser levado em conta na elaboração de projetos de arquitetura de sistemas. De início são definidas as decisões que precisam ser tomadas. No caso deste trabalho, tais decisões partem de problemas encontrados na arquitetura atual. Para cada decisão são definidas uma ou mais abordagens que podem ser seguidas como forma a resolver os problemas-chave. No contexto deste trabalho, as abordagens podem ser vistas como soluções alternativas para um determinado problema. Por exemplo, a redução de custos com a infraestrutura de Chat pode ser atingida através do uso de outro webservice ou desenvolvendo uma infraestrutura própria. Para definir qual é a melhor abordagem, são estabelecidos critérios sobre os quais as abordagens serão confrontadas. Os critérios são restrições da organização em relação à solução a ser desenvolvida; por exemplo, tempo de implementação, uso de tecnologias desconhecidas à equipe de desenvolvimento, etc. Cruzando-se as abordagens com os critérios é possível fazer uma análise mais sistemática e objetiva sobre qual caminho deve ser tomado.

Como subproduto da análise proposta será construída a tabela de decisão que explana as restrições, alternativas, premissas e implicações de cada decisão. Esta tabela serve como instrumento para difundir as razões de cada decisão entre os engenheiros da organização. Além disso servirá como insumo para as próximas fases do ciclo de desenvolvimento da solução.

4.1 Problemas-chave

Conforme explanado na Seção 1.1 foram identificados problemas e requisitos que a implementação atual do sistema de Chat não cobre. A saber:

1) Alto custo financeiro por usuário;
2) Quantidade excessiva de requisições relativas ao Chat feitas para o servidor da aplicação;
3) Solução atual não permite uso off-line

Os pontos 1 e 2 são problemas decorrentes de más decisões de projeto da arquitetura anterior. Já o ponto 3 se trata de um novo requisito que o produto deverá possuir.

4.1.1 Problema 1: Desempenho insatisfatório

Ao realizar *login* no Redu são necessárias $2n + 2$ requisições HTTP, onde n é o número de contatos do usuário, para realizar a autenticação nos canais de Chat. Foi mostrado que a autenticação gera um número que chega a 63% do total de requisições recebidos pelo servidor da aplicação em um mês (ver Seção 1.1.1). Como há um limite 6 para o número de requisições concorrentes permitido nos navegadores modernos ("Browserscope", [S.d.]), detectou-se que estas requisições geram o gargalo de desempenho. Na Seção 1.1.1 é explicado a fundo a origem deste problema.

Para este problema, as seguintes abordagens foram previstas (Tabela 4-1):

#	Abordagem
A3	Otimizar o número de requisições necessárias para a autenticação nos canais do Chat
A4	Alocar num novo servidor, diferente do servidor da aplicação, os módulos responsáveis por receber as requisições de autenticação.

Tabela 4-1: abordagens para melhoria no desempenho

4.1.2 Problema 2: Altos custos por usuário

No momento em que este relatório foi redigido, o *webservice* Pusher, utilizado oferecer a funcionalidade de Chat, realiza a cobrança segundo duas variáveis: número máximo de conexões simultâneas e número de mensagens enviadas. Os custos gerados por tais serviços precisam ser repassados para o usuário final e têm impacto no preço por aluno matriculado no Redu.

Dentro desta restrição, duas abordagens são possíveis, ver Tabela 4-2:

#	Abordagem
A1	Utilização de outro *webservice*, com custo inferior, para realizar as notificações servidor-cliente.
A2	Desenvolver e manter solução própria para notificações servidor-cliente integrada a base de dados do Redu

Tabela 4-2: abordagens para diminuição de custos

4.1.3 Requisito 1: Como tornar possível o uso de Chat no Redu off-line?

Em alguns casos de uso do Redu faz-se necessário que o mesmo funcione em comunidades remotas nas quais o acesso à Internet é nulo ou limitado. Para tal cenário existe uma versão da plataforma que poderá ser executada no servidor de uma LAN sem aceso à Internet. Este servidor proverá uma versão limitada do Redu para os clientes conectados a LAN. Uma vez que o acesso à Internet é nulo ou limitado a alguns períodos do dia as principais funcionalidades não poderão depender de *webservices* disponibilizados fora da LAN.

Para satisfazer este requisito pensou-se em duas abordagens (Tabela 4-3):

#	Abordagem
A2	Desenvolver e manter solução própria para notificações servidor-cliente integrada a base de dados do Redu
A4	Uso de software livre ou proprietário que possa ser executado dentro do servidor da aplicação e seja capaz de substituir o Pusher como canal de comunicação servidor-cliente em tempo real.

Tabela 4-3: abordagens para uso off-line

4.2 Critérios

Para adotar uma entre várias abordagens como a mais apropriada, as mesmas devem ser confrontadas com critérios bem definidos. Estes critérios são baseados em restrições da organização e do projeto como um todo.

4.2.1 Critério 1: Pode ser desenvolvida em até 60 dias?

Este critério representa uma restrição existente no contexto da organização que irá implementar a nova infraestrutura de Chat. Uma vez que a solução visa resolver um problema detectado e enfrentados pelos usuários do sistema, a mesma deve ser implementada num curto período de tempo.

4.2.2 Critério 2: Reduz de maneira efetiva os custos?

Uma vez que o critério custo foi indicado na Seção 1.1.2 como um fator motivador para a remodelagem do serviço de Chat, o mesmo é convertido para um requisito. O termo efetivo, aqui, é relativo aos custos de manutenção e infraestrutura gerados pelo Chat. Para fins de comparação, o custo deverá ser calculado para um cenário onde até 5000 usuários poderão permanecer on-line em um dado momento.

4.2.3 Critério 3: Tecnologia, linguagem e padrões conhecidos pela equie

Este critério representa uma restrição existente no contexto da organização que irá implementar a nova infraestrutura de Chat. Uma vez que a manutenção do sistema proposto será realizada por engenheiros de software que atualmente fazem parte de organização, faz-se necessário que os mesmo tenham um mínimo de domínio sobre a tecnologia utilizada.

4.2.4 Critério 4: Usa tecnologias e serviços testados em produção?

Este critério serve para medir o quão madura certa tecnologia é e se existem casos de uso da mesma em produção.

4.2.5 Critério 5: Melhorar a experiência do usuário no uso do Chat?

Uma vez que a remodelagem do serviço de Chat foi motivado por problemas na experiência de uso do software, este critério deve ser levado em consideração na escolha da abordagem.

4.2.6 Critério 6: Facilita a manutenção

Manutenabilidade da base de código.

4.2.7 Critério 7: Torna a aplicação como um todo mais fácil de escalar?

Este critério é altamente relacionado com o critério 3. A equipe da organização deve conhecer a tecnologia e conceitos envolvidos para que sejam capazes de detectar e resolver os gargalos de desempenho. Por outro lado, a solução escolhida deve possuir estratégias pré-definidas para resolver alguns dos gargalos mais comuns.

4.2.8 Critério 8: Permite uso off-line?

Relacionado ao requisito de uso off-line do Chat

4.3 Decisões

4.3.1 Redução da quantidade de requisições relativas ao Chat

Ao confrontar os critérios com as abordagens A3 e A4 (ver Tabela 4-4), percebe-se que ambas abordagens satisfazem quase todos os critérios. O único critério no qual elas diferem é o C5. Este é decisivo para a escolha da abordagem 3, pois ela é a única que resolve o gargalo de desempenho (no cliente) discutido na Seção 1.1.1). A abordagem A3 também possui um ótimo custo-benefício uma vez que poucas alterações serão realizadas nas regras de negócio e não há necessidade de uso de nenhum serviço/tecnologia desconhecido pela equipe de desenvolimento. Além disso o tempo de implementação é muito inferior ao da abordagem A4. A diminuição das requisições para um número constante, independente da quantidade de contatos, atacaria de frente a limitação detectada na Seção 1.1.1.

Como reduzir a quantidade de requisições relativas ao Chat feitas ao servidor da aplicação?			
#	Critério	A3	A4
C1	Pode ser desenvolvida em até 60 dias?	Sim	Sim
C2	Reduz de maneira efetiva os custos?	Não	Não
C3	Utiliza tecnologias, linguagens e padrões com os quais a equipe de desenvolvimento está familiarizada?	Sim	Sim
C4	Usa tecnologias/serviços testados em produção?	Sim	Sim
C5	Melhora a experiência do usuário no uso do Chat?	Sim	Não
C6	Facilita a manutenção?	Sim	Sim
C7	Facilita a criação de estratégia de escalabilidade?	N/A	N/A
C8	Permite uso off-line?	Não	Não

Tabela 4-4: critérios para reduzir requisições relativas ao chat

Os critérios relativos a redução de custo e uso off-line não são satisfeitos pela abordagem A3. Em relação a redução de custos, esta abordagem não torna o sistema independente do *webservice* Pusher. O Pusher continuaria presente pois seriam necessárias as notificações via WebSocket que nada têm a ver com a requisições de autenticação (feitas via HTTP para o servidor da aplicação). A mesma justificativa serve para explicar a não aceitação pelo critério de uso off-line.

A decisão de arquitetura descrita nesta Seção está representada na Tabela 4-5.

Otimizar o número de requisições necessárias para a autenticação nos canais do Chat	
Problema	Desempenho client-side insatisfatório devido ao grande número de requisições HTTP concorrentes realizadas para a autenticação nos canais de chat.
Decisão	• Estender a API client-side do Pusher para que seja possível realizar a autenticação em vários canais em uma única requisição; • Estender o ChatController (ver Figura 3-2) para aceitar requisições de autenticação em múltiplos canais;
Posições	• Alocar num novo servidor, diferente do servidor da aplicação, os módulos responsáveis por receber as requisições de autenticação.
Limitações	• Não reduz o custo por usuário; • Não permite uso off-line;
Argumentos	Realizar a autenticação de vários canais numa única requisição HTTP irá remover o gargalo de desempenho quanto ao número de requisições concorrentes. Esta decisão pode ser implementada rapidamente e resolve um problema crítico que atinge o usuário final.

Tabela 4-5: decisão de projeto

4.3.3 Redução de custos

No tocante a redução de custos, foram consideradas nas abordagens A1 e A2. A abordagem A1 tenta atacar o problema utilizando outro *webservice* de notificações. Tal abordagem foi descartada pois um *benchmarking* de custos foi realizado e o Pusher se mostrou como alternativa com o menor custo no contexto atual.

Como reduzir os custos gerados pelo Chat?			
#	Critério	A1	A2
C1	Pode ser desenvolvida em até 60 dias?	Sim	Sim
C2	Reduz de maneira efetiva os custos?	Não	Sim
C3	Utiliza tecnologias, linguagens e padrões com os quais a equipe de desenvolvimento está familiarizada?	Sim	Sim
C4	Usa tecnologias/serviços testados em produção?	Sim	Sim
C5	Melhora a experiência do usuário no uso do Chat?	N/A	N/A
C6	Facilita a manutenção?	Não	Sim
C7	Facilita a criação de estratégia de escalabilidade?	Não	Sim
C8	Permite uso off-line?	Não	Sim

A abordagem A2 (ver Figura 4-1) propõe o projeto e implementação de um serviço hospedado que tem como objetivo realizar as notificações servidor-cliente. Tal serviço deve ser mantido pela equipe do Redu e poderia ser aproveitado para outras aplicações em tempo real. Esta alternativa mostrou-se ótima pois ataca dois problemas críticos: custos altos (C2) e o requisito para uso off-line (C8). O uso off-line é possível por se tratar de um serviço hospedado, ou seja, que irá funcionar dentro LAN que se encontra os servidores da aplicação. Logo há a possibilidade de comunicação entre cliente, Chat e aplicação sem necessidade de acesso à Internet.

Figura 4-1: abordagens para redução de custos

A decisão de projeto para este problema-chave é apresentada na Tabela 4-6

Desenvolver solução própria para notificações servidor-cliente	
Problema	O uso do *webservice* Pusher gera altos custos financeiros por usuário. Tal fato se reflete no preço do Redu que, por sua vez, aumenta as barreiras de entrada para usuários.
Decisão	Criar *webservice* próprio responsável por realizar as notificações servidor-client necessária na funcionalidade de Chat.
Alternativas	• Utilizar outro *webservice* com custos inferiores ao Pusher
Limitações	N/A
Argumentos	Custos de manutenção de um servidor para o *webservice* privado de Chat são inferiores ao custo do *webservice* Pusher

Tabela 4-6: decisão de projeto

5 Arquitetura proposta

Neste capítulo será apresentada uma proposta de arquitetura para a nova versão do serviço de Chat. Tal arquitetura é resultado das decisões de projeto definidas no capítulo anterior.

5.1 Entrada e saída assíncronos

O servidor de Chat é construído utilizando o padrão de projeto conhecido com Reator (COPLIEN *et al.*, 1995). Este padrão torna possível realizar ações de entrada e saída (I/O) de forma não bloqueante e sem necessidade de criação e manutenção de threads. Esta decisão foi motivada pela necessidade de atender muitas requisições concorrentes e devido aos poucos recursos de infraestrutura disponíveis.

O elemento principal do padrão de projeto Reator (reactor pattern) é o reator propriamente dito. O reator, em poucas palavras, é um loop que é responsável por criar e executar descritores de sistema operacional (*descriptors*). *Descriptors* são elementos dos sistemas operacionais que servem como uma abstração para acesso a arquivos e sockets. Os *descriptors* são a chave da característica não bloqueante deste sistema pois, no momento de criação, é possível dizer para o programa não bloquear em ações de entrada ou saída. Por exemplo, ao criar-se um *descriptor* para um socket é possível dizer que ele não deve bloquear a execução e, quando houver entrada de dados, os mesmo deverão ser adicionados a um buffer.

O processo de buffering e checagem de entradas é feito pelo sistema operacional através de chamadas ao sistema (a mais comum é a select()). Existem outras chamadas de sistema com o mesmo propósito. Por exemplo, em sistemas Linux existe o epoll e no BSDs o kqueue.

De um ponto de vista mais abstrato, o reator é responsável por informar a aplicação quando dados estão disponíveis num determinado descritor. A abstração utilizada para tal é a de eventos, ou seja, eventos ocorrem em determinada situações e a aplicação deve criar handlers que respondam aos mesmo.

5.2 Componentes do sistema

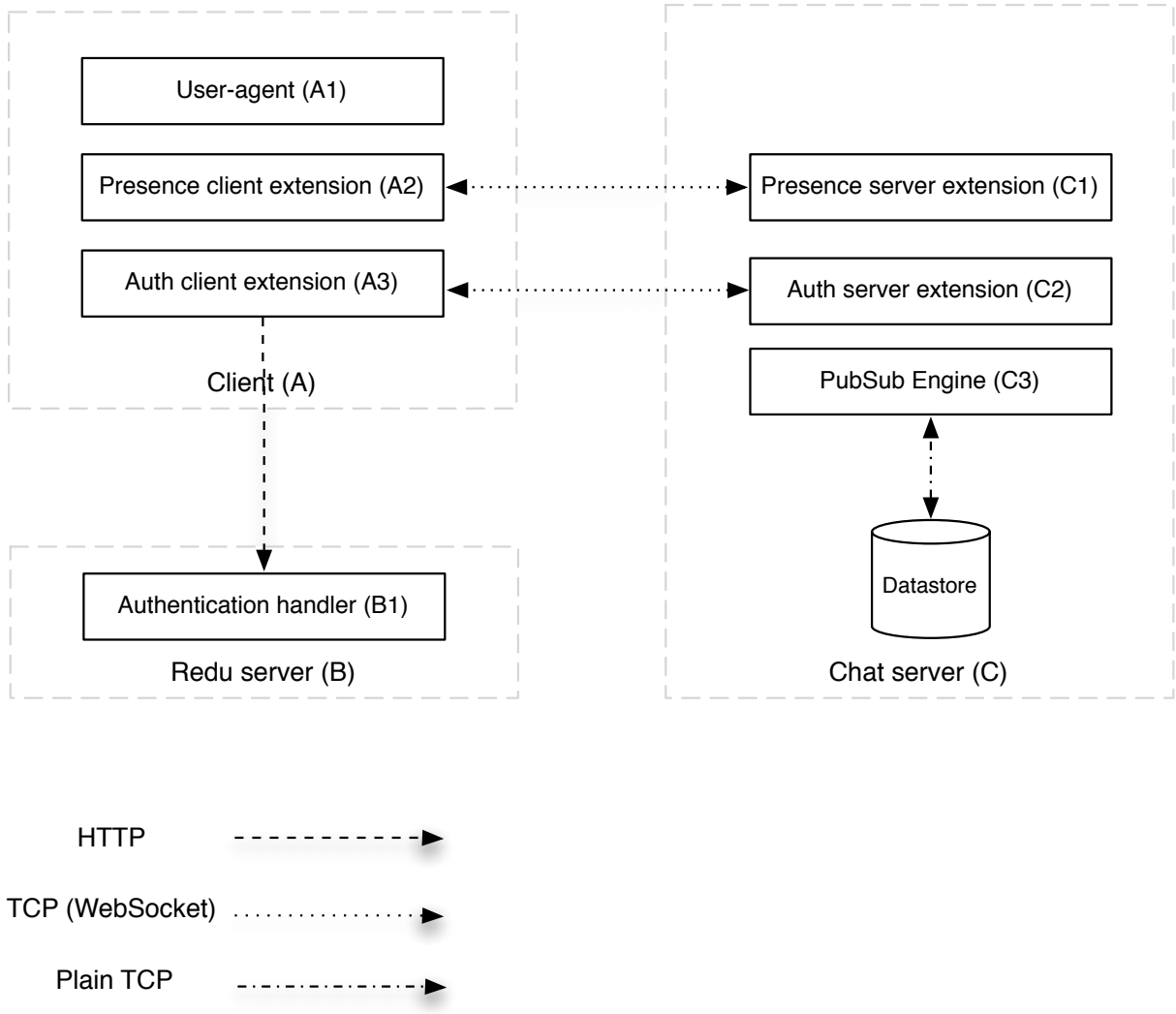

Figura 5-1: componentes da arquitetura proposta

Em relação a arquitetura anterior esta nova arquitetura se difere por tratar o envio e recebimento de mensagens de Chat dentro de um novo servidor (C, na Figura 5-1). O servidor da aplicação (B), que outrora participava de todas as interações, se limita a criar credenciais para que o cliente (A) seja capaz de enviar e receber mensagens de forma privada.

No diagrama da Figura 5-1 são mostrados os componentes do sistema divididos por hosts que fazem parte da aplicação de Chat. Cada host é representado por uma caixa tracejada. O host cliente (A) é uma representação da máquina do usuário que interage com os demais host através de um navegador. O host Redu (B) se trata de uma representação do servidor da aplicação. O host Servidor de Chat (C) representa o servidor responsável pelo recebimento e envio de mensagens de Chat entre vários clientes conectados a aplicação. Cada host

está num domínio diferente e interagem entre sí através de requisições HTTP ou TCP (através de WebSockets). O componente agente de usuário

Do ponto de vista do agente do usuário, criou-se uma abstração para o envio e recebimento de mensagens. Esta abstração foi feita segundo o padrão de comunicação Publisher-Subscriber e foi justificada pois torna mais simples a divisão entre mensagens de presença (on-line e off-line) e outros tipos de mensagens (mensagem de Chat, por exemplo). Além disso torna mais fácil a criação de outras aplicações que utilizem a mesma infraestrutura de Chat para receber e enviar notificações. Esta abstração já era utilizada na solução anterior e poucas mudanças na API do agente do usuário foram necessárias.

Este componente permite a criação de extensões, que são um maneira de interceptar mensagens para a adição de novas informações às mesmas. No caso da aplicação de Chat, tais extensões são utilizadas para injetar credenciais e informações de presença dentro das mensagens.

A API do agente de usuário é escrita em JavaScript e se utiliza de callbacks para reagir a eventos (mensagens enviadas do servidor de Chat). O uso do JavaScript é motivado por ser uma linguagem padrão e amplamente interpretada dentre os navegadores modernos.

5.2.1 Extensão de presença

A extensão de presença (A2) é responsável por reagir a mensagens de presença enviadas do servidor de Chat (C). Tais mensagens de presença são responsáveis por informar o status (on-line/off-line) de um determinado usuário. Esta extensão também reage a eventos de presença. Um exemplo seria adicionar um usuário a lista de contatos quando o mesmo está on-line.

5.2.2 Extensão de autenticação

A extensão de autenticação (A3) é responsável por injetar credenciais dentro de todas mensagens enviadas para o servidor de Chat. Tais credenciais são uma forma de garantir que somente clientes com a devida autorização serão capazes de enviar e receber mensagens num determinado canal. Tais credenciais são geradas pelo compnente *authentication handler* (B1).

5.2.3 Controlador de autenticação

Este componente, antes o principal do sistema, teve sua responsabilidade reduzida a criar credenciais para usuários (autorização) e fornecer informações a respeito de um ou mais usuários. Ambas interações acontecem através de uma API REST e são direcionados ao servidor da aplicação Redu.

Conforme detectado na Seção 1.1.1 o fato de as requisições serem feitas para a aplicação Redu não caracterizam um gargalo desde que não sejam feitas em grande quantidade e em paralelo. Este componente sofreu uma alteração na sua API para que não sejam necessárias mais de duas requisições HTTP para realizar a autenticação nos canais de todos os contatos. Este número é constante e independe do número de contatos. Desta forma o gargalo de excesso de requisições é resolvido.

O componente Redu passou a ser responsável apenas pelas regras de negócio de autenticação e leitura de dados do usuário. Esta lógica não foi repassada para o servidor do Chat (C) pois não se apresenta como um gargalo de performance e por fazer parte da responsabilidade (SHALLOWAY; TROTT, 2004) do componente Redu.

5.2.4 Servidor de Chat

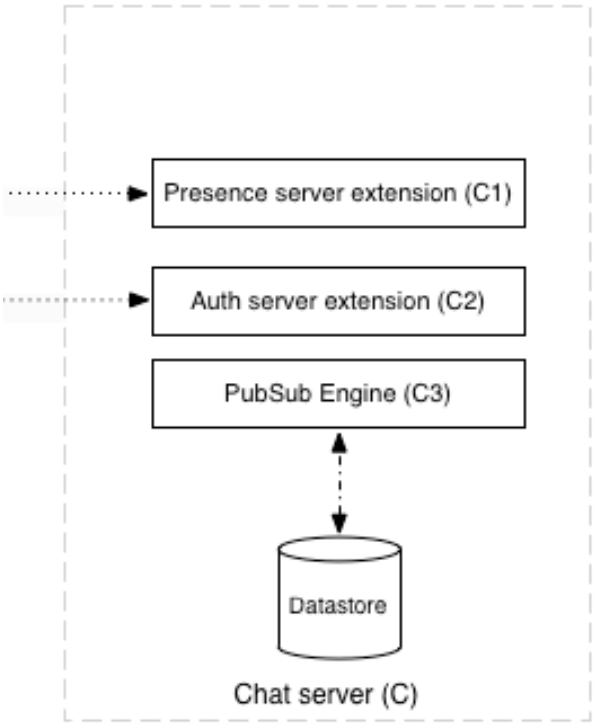

Figura 5-2: servidor de chat

O servidor de Chat substitui o *webservice* Pusher e, além disso, é responsável por receber mensagens diretamente dos clientes conectados. Ou seja, sem necessidade de passar pelo componente Redu.

O servidor de Chat, conforme dito na Seção 5.1 implementa uma arquitetura orientada a eventos e não bloqueante para ações de entrada e saída baseada no padrão de projeto Reator. A passagem de mensagens entre dois ou mais clientes

de Chat são feitas através deste servidor usando-se da abstração Publisher-subscriber.

A passagem de mensagens é realizada pelo componente PubSub Engine (C3). Este componente é uma implementação do especificação de mensagens Bayeux (ZHUANG *et al.*, 2001). O Bayeux é amplamente utilizado em aplicações publisher-subscriber Web e define como a conexão entre cliente e servidor deve acontecer, bem como o procedimento para a passagem de mensagens. Esta especificação é agnóstica de protocolo, ou seja, é capaz de ser implementada tanto sobre HTTP quanto sobre WebSockets. Além disso a especificação dá possibilidade a criação de extensões do protocolo. Estas extensões, dentro da aplicação de Chat, foram usadas para adicionar funcionalidades de presença e autenticação.

O PubSub Engine (C3) provê uma camada de abstração sobre o armazenamento utilizado. Informações com o registro de clientes conectados e clientes inscritos num canal são armazenadas no Datastore.

A validação de credenciais é realizada pela extensão de autenticação (C2). Esta extensão se limita a procurar por credenciais nas mensagens enviadas pelos clientes. Estas credenciais são validadas junto a um token secreto compartilhado com o servidor do Redu (B na Figura 5-1). Os detalhes da validação de credenciais serão explorados na Seção 5.3.

Mensagens de presença são enviadas para os clientes conectados através da extensão de presença (C2). Esta extensão é responsável por ouvir eventos de conexão e desconexão de clientes informar aos interessados que este cliente está on-line ou ficou off-line.

A lógica do padrão Publisher-Subscriber é gerenciada um SGBD não relacional que implementa tal padrão. No servidor de Chat é utilizado o Eventmachine (GUPTA, 2011). O Evenmachine é uma biblioteca escrita em Ruby (MATSUMOTO, 2011) e C++ responsável pela criação e execução dos descriptors do sistema operacional. Esta escolha foi justificada pois a equipe de desenvolvimento possui fluência em Ruby. Além disso a linguagem oferece facilidade na criação de DSLs (domain specific languagens) que tornam mais simples a criação de abstrações. Uma vez que aplicações orientadas a eventos podem facilmente se tornar complexas e difíceis de manter, tais abstrações se tornam importante. O componente PubSub engine e a implementação do protocolo Bayeux é realizada por uma biblioteca do EventMachine chamada Faye (COLGAN, 2009).

5.3 Autenticação

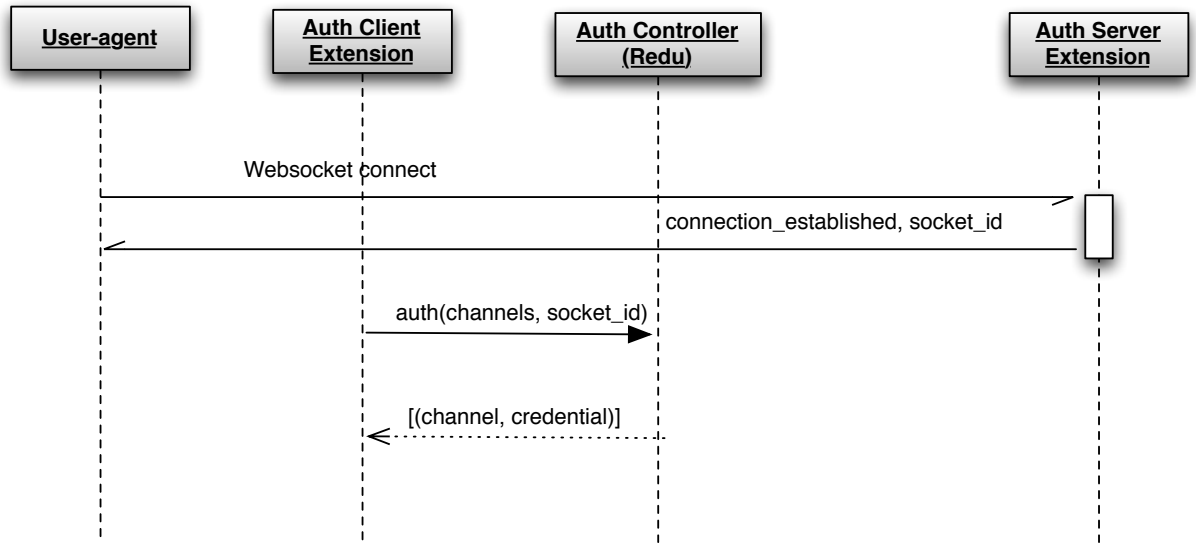

Figura 5-3: diagrama de sequência para autenticação (proposta)

O procedimento de autenticação garante que somente usuários autorizados têm a capacidade de publicar ou se inscrever em canais. Este controle é feito por credenciais geradas a partir da combinação do ID do usuário, canal conectado e uma chave secreta armazenada no servidor do Redu e no servidor de Chat. A criação das credenciais é feita conforme o descrito na Seção 3.4.

Quando o navegador completa o carregamento de uma determinada página do Redu, o componente agente do usuário carrega os componentes responsáveis pelo Chat. Quando o carregamento é finalizado, a extensão de autenticação realiza a requisição de credenciais via HTTP para o Redu (Auth controller). O Redu gera as credenciais e as envia de volta para a extensão de autenticação. Ao receber a resposta, a extensão de autenticação já é capaz de enviar mensagens credenciadas para o servidor de Chat. Todas as mensagens recebidas pelo servidor de Chat passam pela extensão de autenticação do servidor, que verifica a existência de credenciais e às valida.

5.4 Passagem de mensagens

Após a autenticação o servidor do Redu não participa mais da passagem de mensagens. Todas as mensagens são enviadas via WebSocket diretamente para o servidor de Chat que, por sua vez, às repassa para os receptores. Conforme o diagrama de sequência da Figura 5-4, o componente de autenticação (cliente) injeta as credenciais nas mensagens publicadas. Tais credenciais são validadas pelo componente de autenticação (servidor) e, caso sejam válidas, são repassadas para o componente de publicação (PubSub Engine). A fase final desta sequência é

o recebimento da mensagem por todos os clientes conectados ao servidor de Chat. Utilizando-se a API do componente agente de usuário callbacks podem ser registrados para reagir ao recebimento de certas mensagens.

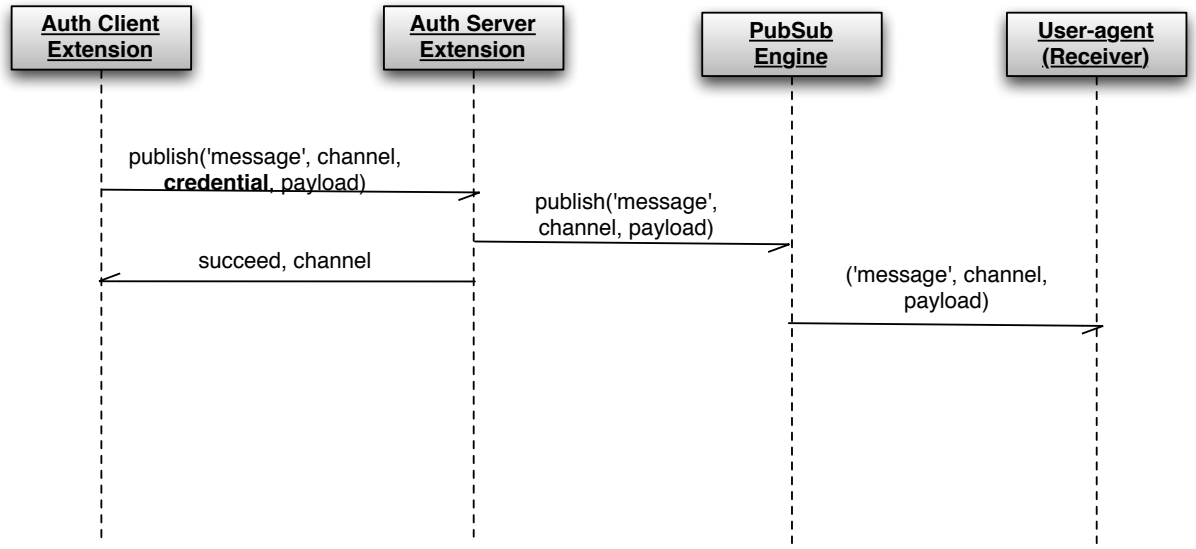

Figura 5-4: diagrama de sequência para passagem de mensagem (proposta)

6 Resultados

No desenvolvimento deste trabalho foram identificados problemas-chave na funcionalidade de Chat do produto Redu. No Capítulo 4 abordagens alternativas foram enumeradas e escolhidas tomando-se como base premissas (critérios). No Capítulo 5, uma arquitetura do sistema foi redesenhada em conformidade com as decisões de projeto. Neste Capítulo serão apresentados os resultados e subprodutos deste trabalho.

6.1 Problema de performance

O problema de desempenho descrito na Seção X foi gerado pelo número excessivo de requisições concorrentes realizadas pelo ChatClient (ver Figura X). A abordagem mais apropriada para resolver este problema foi "Otimizar o número de requisições necessárias para a autenticação nos canais do Chat" (ver Seção X). Esta abordagem foi implementada com durante o desenvolvimento deste trabalho e se mostrou efetiva na solução do problema de desempenho. A solução implementada realiza a autenticação com um número constante de requisições. Ou seja, independente do número de canais aos quais o agente do usuário se conecta, apenas duas requisições HTTP serão realizadas para o servidor da aplicação.

Analisando-se quantitativamente na Tabela 6-1 apresentada na Seção 1.1.1 e replicada abaixo, são mostrados o número de requisições HTTP por controlador na aplicação Redu durante o período de um mês. Conforme mostrado, o controlador responsável pela autenticação de usuários nos canais de Chat (ChatController#auth) recebe 67% do total de requisições da aplicação. Este dado foi o ponto de partida para a detecção de um gargalo de performance relativo ao número de requisições concorrentes que o navegador do cliente é capaz de realizar.

Controlador	Hits	% do Total
ChatController#auth	156145	63.7%
EnvironmentsController#preview	32221	13.2%
UsersController#home	5834	2.4%
BaseController#site_index	5282	2.4%
CoursesController#show	3301	1.3%

Tabela 6-1: requisições por controlador

Na Tabela 6-2 são mostrados o número de requisições após a implementação da solução intermediária. O controlador ChatController#multiauth realiza a autenticação em múltiplos canais em uma única requisição. Este controlador, antes responsável por 63,7% das requisições, agora recebe apenas 19,2% do total de requisições em um mês.

Controlador	Hits	% do Total
ChatController#multiauth	16989	19.2%
UsersController#index	9452	10.7%
BaseController#site_index	6806	7.7%
UsersController#home	6562	7.4%
EnvironmentsController#show	2799	3.2%

Tabela 6-2: requisições por controlador pós otimizações

Analisando os dados do controlador de autenticação antes e depois da otimização também nota-se uma melhora significativa no tempo médio de resposta (ver Tabela 6-2). A requisição de autenticação antes da otimização levava, em média, 0,93 segundos para ser executada. Após a implementação da autenticação em multiplos canais numa mesma requisição (ChatController#multiauth) este número caiu para, em média 0,4 segundos.

Controlador	Média	Desvio padrão
ChatController#auth	0.93s	1.27s
ChatController#multiauth	0.40s	0.63s

Tabela 6-3: tempo médio por requisição pós otimizações

6.2 Redução de custos e uso off-line

Na Seção 1.1.2 foi identificado um custo financeiro elevado gerado pela funcionalidade de Chat. Este custo, em sua maior parte, se deve ao uso do *webservice* Pusher. A abordagem mais apropriada para resolver este problema foi o "Desenvolvimento de solução própria para notificações servidor-cliente" (ver Seção 4.3.2).

Uma prova de conceito desta abordagem foi implementada durante o desenvolvimento deste projeto. Todos os componentes apresentados na diagrama, Figura 6-1 foram implementados com exceção dos componentes PresenceServerExtension e PresenceClientExtension, representados em branco. O serviço de Chat desenvolvido foi instalado no servidor de homologação juntamente com a aplicação e apresentou resultados satisfatórios.

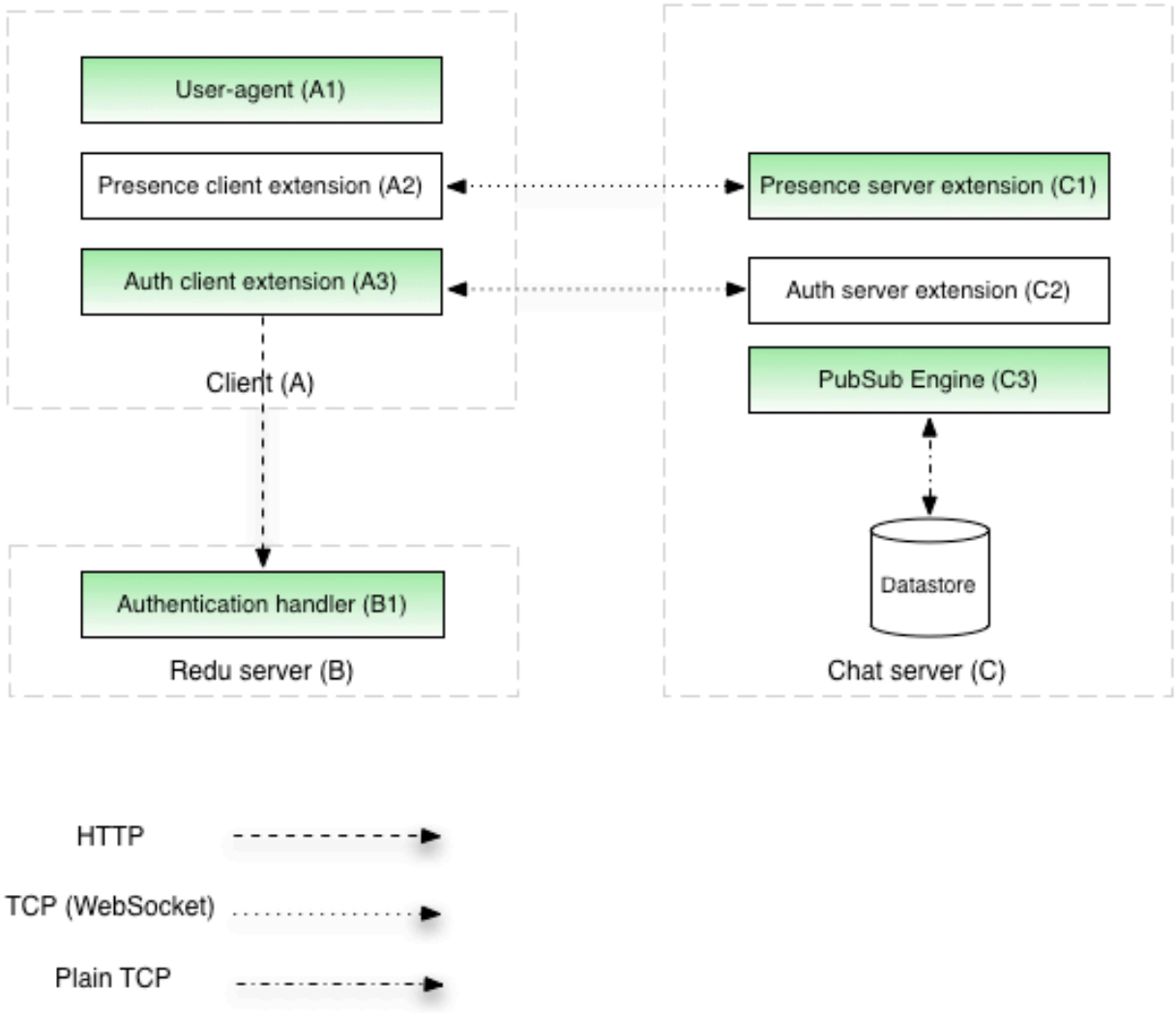

Figura 6-1: componentes implementados na prova de conceito

Também em ambiente de homologação foi comprovado que o serviço de Chat desenvolvido é capaz de funcionar em modo off-line, ou seja, sem conexão com a Internet. Esta funcionalidade satisfaz o requisito apresentado na Seção 1.1.3 que permitirá o uso do Redu em comunidades remotas com acesso restrito a Internet.

7 Conclusão e trabalhos futuros

O uso de diálogo síncrono, ou seja, em tempo real, em ambientes virtuais de aprendizagem te um impacto positivo na redução da distância transacional entre aprendizes e professores. O desenvolvimento deste trabalho possibilitou a redução de custos financeiros e a melhoria da experiência do usuário no uso de uma funcionalidade de Chat em uma Rede Social Educacional. Tais melhorias contribuíram para o aumento da competitividade de um produto real através da diminuição das barreiras de entrada e o aumento da sinergia entre mercado e academia.

7.1 Trabalhos futuros

7.1.1 Generalização para aplicações em tempo real

Dentro da Rede Social Educacional Redu existem outras funcionalidades que são intrinsicamente assíncronas. Atualmente estas funcionalidades fazem uso do Pusher ou de outras técnicas para simular seu comportamento. Um próximo passo seria criar um serviço interno único e generalizado capaz de dar suporte a qualquer funcionalidade em tempo real.

7.1.2 Informações de presença

A prova de conceito implementada não estabelece uma maneira simples de representar informações de presença (on-line/off-line) dos usuários conectados aos canais. A necessidade deste tipo de informação vai bem além da funcionalidade de Chat e poderia ser utilizada para criar informações de presença de forma contextual. Por exemplo, seria possível incentivar a colaboração e o diálogo mostrando quais usuários da rede social estão visualizando uma determinada aula num dado momento.

8 Bibliografia

- Browserscope. Disponível em: <http://www.browserscope.org/?category=network&v=top>. Acesso em: 11 dez. 2011.
- Censo ead.br. . [S.l.]: Pearson Education do Brasil. , 2010
- COLGAN, J. Faye: Simple pub/sub messaging for the web. Disponível em: <http://faye.jcoglan.com/>. Acesso em: 13 dez. 2011.
- COPLIEN, E. J.; SCHMIDT, D. C.; SCHMIDT, D. C. Reactor - An Object Behavioral Pattern for Demultiplexing and Dispatching Handles for Synchronous Events. 1995.
- FIELDING, R. Architectural Styles and the Design of Network-based Software Architectures. Disponível em: <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>. Acesso em: 30 nov. 2011.
- GARRET, J. Ajax: A New Approach to Web Applications - Adaptive Path. Disponível em: <http://adaptivepath.com/ideas/ajax-new-approach-web-applications>. Acesso em: 30 nov. 2011.
- GUPTA, A. eventmachine @ GitHub. Disponível em: <http://rubyeventmachine.com/>. Acesso em: 1 dez. 2011.
- MATSUMOTO, Y. Ruby Programming Language. Disponível em: <http://www.ruby-lang.org/en/>. Acesso em: 1 dez. 2011.
- MELO, C. Scaffolding of Self-Regulated Learning in Social Networks. . [S.l: s.n.]. , 2010
- MOORE, M. G. Handbook of distance education. [S.l.]: Routledge, 2007.
- PUSHER. Pusher | HTML5 WebSocket Powered Realtime Messaging Service. Disponível em: <http://pusher.com/>. Acesso em: 30 nov. 2011.

- SHALLOWAY, A.; TROTT, J. **Design Patterns Explained: A New Perspective on Object-Oriented Design (2nd Edition) (Software Patterns Series)**. [S.l.]: Addison-Wesley Professional, 2004.
- **The WebSocket API**. , The WebSocket API. [S.l: s.n.]. Disponível em: <http://dev.w3.org/html5/websockets/>. Acesso em: 30 nov. 2011. , 2011
- TYREE, J.; AKERMAN, A. Architecture decisions: Demystifying architecture. **Software, IEEE**, v. 22, n. 2, p. 19–27, 2005.
- ZHUANG, S. Q.; ZHAO, B. Y.; JOSEPH, A. D.; KATZ, R. H.; KUBIATOWICZ, J. D. **Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination**. Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video. **Anais**..., NOSSDAV '01. New York, NY, USA: ACM. Disponível em: <http://doi.acm.org/10.1145/378344.378347>. Acesso em: 13 dez. 2011. , 2001