UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

VINÍCIUS BULHÕES DA SILVA LIMA

Middleware para Migração de Sessões de Interação de Aplicações Web

Suporte para o desenvolvimento de aplicações Web migráveis

Goiânia
2017

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TESES E DISSERTAÇÕES NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), regulamentada pela Resolução CEPEC nº 832/2007, sem ressarcimento dos direitos autorais, de acordo com a Lei nº 9610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou *download*, a título de divulgação da produção científica brasileira, a partir desta data.

1. Identificação do material bibliográfico: **[X] Dissertação** **[ ] Tese**

2. Identificação da Tese ou Dissertação:

Nome completo do autor: Vinícius Bulhões da Silva Lima

Título do trabalho: **Middleware para Migração de Sessões de Interação de Aplicações Web: Suporte para o desenvolvimento de aplicações Web migráveis**

3. Informações de acesso ao documento:

Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

Havendo concordância com a disponibilização eletrônica, torna-se imprescindível o envio do(s) arquivo(s) em formato digital PDF da tese ou dissertação.

Assinatura do(a) autor(a)[2]

Ciente e de acordo:

Assinatura do(a) orientador(a)[2]

Data: 07 / 02 / 2018

[1]Neste caso o documento será embargado por até um ano a partir da data de defesa. A extensão deste prazo suscita justificativa junto à coordenação do curso. Os dados do documento não serão disponibilizados durante o período de embargo.

Casos de embargo:
- Solicitação de registro de patente
- Submissão de artigo em revista científica
- Publicação como capítulo de livro
- Publicação da dissertação/tese em livro

[2]A assinatura deve ser escaneada.

VINÍCIUS BULHÕES DA SILVA LIMA

Middleware para Migração de Sessões de Interação de Aplicações Web

Suporte para o desenvolvimento de aplicações Web migráveis

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação.

Orientador: Prof. Dr. Fábio Moreira Costa

Goiânia
2017

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Bulhões da Silva Lima, Vinícius
Middleware para Migração de Sessões de Interação de Aplicações Web [manuscrito] : Suporte para o desenvolvimento de aplicações Web migráveis / Vinícius Bulhões da Silva Lima. - 2017.
LXIV, 64 f.

Orientador: Prof. Dr. Fábio Moreira Costa.
Dissertação (Mestrado) - Universidade Federal de Goiás, Instituto de Informática (INF), Programa de Pós-Graduação em Ciência da Computação, Goiânia, 2017.
Bibliografia.
Inclui lista de figuras.

1. middleware. 2. comportamento follow-me. 3. aplicações web. 4. migração de sessão. I. Moreira Costa, Fábio, orient. II. Título.

CDU 004

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UFG

ATA Nº 31/2017

ATA DA SESSÃO DE JULGAMENTO DA DISSERTAÇÃO DE MESTRADO DE VINÍCIUS BULHÕES DA SILVA LIMA

Aos vinte e cinco dias do mês de outubro de dois mil e dezessete, às oito horas e trinta minutos, na sala 150 do Instituto de Informática da Universidade Federal de Goiás, Campus Samambaia, reuniu-se a banca examinadora designada na forma regimental pela Coordenação do Curso para julgar a dissertação de mestrado intitulada **"Middleware para Migração de Sessões de Interação de Aplicações Web: Suporte para o desenvolvimento de aplicações Web migráveis"**, apresentada pelo aluno Vinícius Bulhões da Silva Lima como parte dos requisitos necessários à obtenção do grau de Mestre em Ciência da Computação, área de concentração Ciência da Computação. A banca examinadora foi presidida pelo orientador do trabalho de dissertação, Professor Doutor Fábio Moreira Costa (INF/UFG), tendo como membros os Professores Doutores Bruno Oliveira Silvestre (INF/UFG) e Arlindo Flávio da Conceição (ICT/UNIFESP). O prof. Arlindo Flávio da Conceição participou à distância por videoconferência. Aberta a sessão, o candidato expôs seu trabalho. Em seguida, o aluno foi arguido pelos membros da banca e:

(X) tendo demonstrado suficiência de conhecimento e capacidade de sistematização do tema de sua dissertação, a banca concluiu pela **aprovação** do candidato, sem restrições.

( ) tendo demonstrado suficiência de conhecimento e capacidade de sistematização do tema de sua dissertação, a banca concluiu pela **aprovação** do candidato, condicionado a satisfazer as exigências listadas na Folha de Modificação de Dissertação de Mestrado anexa à presente ata, no prazo máximo de 30 dias, a contar da presente data, ficando o professor-orientador responsável por atestar o cumprimento dessas exigências.

( ) não tendo demonstrado suficiência de conhecimento e capacidade de sistematização do tema de sua dissertação, a banca concluiu pela **reprovação** do candidato.

Os trabalhos foram encerrados às 12 horas. Nos termos do Regulamento Geral dos Cursos de Pós-Graduação desta Universidade, lavrou-se a presente ata que, lida e julgada conforme, segue assinada pelos membros da banca examinadora.

Prof. Dr. Fábio Moreira Costa _____

Prof. Dr. Bruno Oliveira Silvestre _____

Prof. Dr. Arlindo Flávio da Conceição _____

VINÍCIUS BULHÕES DA SILVA LIMA

Middleware para Migração de Sessões de Interação de Aplicações Web

Suporte para o desenvolvimento de aplicações Web migráveis

Dissertação defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Ciência da Computação, aprovada em 25 de Outubro de 2017, pela Banca Examinadora constituída pelos professores:

Prof. Dr. Fábio Moreira Costa
Instituto de Informática – UFG
Presidente da Banca

Prof. Dr. Bruno Oliveira Silvestre
INF – UFG

Prof. Dr. Arlindo Flavio da Conceição
ICT – UNIFESP

Vinícius Bulhões da Silva Lima

Graduou–se em Ciências da Computação na UFG – Universidade Federal de Goiás. Durante sua graduação, foi pesquisador do CNPq em conjunto com a RNP (Rede Nacional de Ensino e Pesquisa) em um trabalho de iniciação científica no projeto GT-ATER no Instituto de Informática – UFG. Também foi pesquisador da RNP no projeto GT-TeI no Instituto de Informática – UFG em conjunto com a Universidade Federal do Rio de Janeiro (UFRJ) e a Pontifícia Universidade Católica do Rio de Janeiro (PUC–RJ). Durante o Mestrado, na UFG, foi bolsista da CAPES e realizou estágio docência na mesma universidade na qual realizou o mestrado. Atualmente trabalha com aplicações ubíquas.

Dedico este trabalho ao meus familiares e amigos que estiveram comigo desde o início da jornada no mestrado.

Agradecimentos

Agradeço primeiro a Deus por ter me mantido e dado a oportunidade de concluir o mestrado.

Agradeço ao meus familiares e amigos que estiveram comigo durante todo o período do mestrado, em especial minha mãe Lúcia e minha avó Maronilde.

Agradeço ao meu professor orientador Fábio M. Costa pela paciência e disponibilidade para guiar o percurso do trabalho desenvolvido.

Agradeço ao professor David Bromberg e ao seu aluno Adrien Luxey, ambos da Universidade de Rennes - França por contribuírem com os conceitos e implementação realizados no trabalho.

Agradeço a Fundação CAPES por disponibilizar uma bolsa de estudo para o custeio das despesas durante o período do mestrado.

Shoot for the moon. Even if you miss, you'll land among the stars!

Les Brown,
Live Your Dreams.

Resumo

Bulhões da Silva Lima, Vinícius. **Middleware para Migração de Sessões de Interação de Aplicações Web**. Goiânia, 2017. 63p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

O mercado oferece vários dispositivos com capacidade de processamento suficiente para executar aplicações que anteriormente eram apenas suportadas por máquinas mais robustas, como os computadores de mesa. Tais aplicações possuem diferentes versões finais para cada plataforma e sistema operacional, pois os recursos como capacidade de processamento, armazenamento e periféricos mudam e a interação com o usuário varia de acordo com esses recursos. Essa diversidade possibilita usuários trabalharem com a mesma aplicação em diferentes dispositivos, em diferentes locais e durante diferentes períodos do dia. Com essas possibilidades oferecidas ao usuário, as aplicações necessitam de suporte para que os usuários possam facilmente mudar de um dispositivo para outro e continuar suas tarefas no mesmo ponto no qual pararam. As aplicações Web estão crescendo em popularidade entre os usuários e também podem se beneficiar do suporte para mudar o dispositivo no qual interagem com o usuário. Este trabalho apresenta a arquitetura de um *middleware* por meio do qual aplicações Web podem adquirir a capacidade de migrar sessões de interação com o usuário. Como resultado, as sessões de interação das aplicações Web, juntamente com os dados relacionados a elas, podem ser transferidas de um dispositivo para outro de forma automática e transparente. Uma implementação da proposta do *middleware* foi feita a fim de avaliar a abordagem. Também é avaliado o efeito da escolha do mecanismo de transferência para o envido dos dados relacionados a sessão de interação da aplicação.

Palavras–chave

middleware, comportamento follow–me, aplicações web, migração de sessão

Abstract

Bulhões da Silva Lima, Vinícius. **Middleware for Interaction Session Migration of Web Applications**. Goiânia, 2017. 63p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

Nowadays, it is common to find a variety of powerful devices which execute applications that were only supported by machines that were traditionally more capable, such as desktop PCs. Due to the variety of plataforms and operating systems, which exhibit different available resources and user interaction methods, applications need to be provided in different versions. As a result, users work with the same application on different devices, at different locations and at different times of the day. Facing these user's new possibilities, applications need support to allow users to easily switch from one device to another and resume their tasks at the same point where they stopped. This work introduces a middleware architecture that can be used by web applications to acquire the interaction session migration capability. Thus, web applications' interaction sessions, and their related data, can be migrated from one device to another in an automatic and seamless way. A middleware platform was designed and implemented in order to evaluate the proposed approach. Also, it is evaluated the effect of transfer mechanisms on the handoff process of the interaction session application.

Keywords

middleware, follow-me behavior, web application, session migration

Sumário

Lista de Figuras

CAPÍTULO 1

Introdução

Com a popularização de dispositivos como *laptops*, smartphones e tablets, torna-se comum uma única pessoa possuir vários dispositivos com capacidade de processamento suficiente para executar aplicações que anteriormente eram suportadas apenas por máquinas mais robustas como os computadores de mesa (*desktops*). Também é comum uma aplicação possuir diferentes versões finais, uma para cada dispositivo e sistema operacional. Essas múltiplas versões são necessárias para disponibilizar a aplicação nessa variedade de dispositivos, onde os recursos disponíveis são diferentes e a interação com usuário pode, e algumas vezes deve, ser feita de forma diferente. Da mesma forma, é comum usuários trabalharem com a mesma aplicação em diferentes dispositivos, em diferentes locais e durante diferentes períodos do dia.

Entre essa diversidade de aplicações e dispositivos, as aplicações Web estão se tornado cada vez mais populares entre os usuários. Qualquer dispositivo com acesso à Internet e que possua um navegador Web instalado pode facilmente carregar e executar uma aplicação Web. Exemplos de aplicações Web são: as redes sociais, os editores de documento, os jogos on-line, os serviços de pesquisa, entre outros. Aplicações Web executaram sobre uma plataforma homogênea, assim, não sendo necessárias mais de uma versão da mesma aplicação para executar em diferentes dispositivos.

Gerenciar separadamente cada dispositivo, navegador e aplicação Web a fim de migrar de um dispositivo para outro e manter a interação com a aplicação Web pode ser uma tarefa tediosa para o usuário e aumenta os requisitos aos quais os desenvolvedores de aplicações Web devem estar atentos. Com mais de um dispositivo conectado à Web e sobre o qual executam aplicações que consumem os conteúdos e serviços oferecidos por ela, permitir que o usuário mude o dispositivo com o qual está interagindo é uma extensão natural da plataforma Web.

Permitir que o usuário mude de dispositivo enquanto mantém sua sessão de interação com a aplicação Web envolve, coletar o atual estado da sessão de interação da aplicação, enviar esse estado de sessão de um dispositivo para outro através de um mecanismo de transferência e restabelecer a sessão de interação da aplicação. Logo, é necessária uma abordagem para permitir que os usuários possam facilmente mudar de

um dispositivo para outro e continuar sua sessão de interação com a aplicação Web sem interrupção.

1.1 Descrição do problema

A Web provê independência de localização ao acessar dados. O usuário, ao realizar poucas ações, pode, através de seus dispositivos, acessar informações armazenadas remotamente de maneira tão fácil como se elas estivessem armazenadas localmente. Não é mais necessário diferenciar dados armazenados remota e localmente [30], nem mesmo utilizar diferentes ferramentas para acessar cada um deles [9], de fato, o usuário não precisa saber onde o dado está armazenado.

Apesar do grande sucesso, a base da Web ainda é relativamente estática. Ela provê suporte para um conjunto finito de protocolos para mover dados entre dispositivos e cada dispositivo pode apenas manipular os dados de acordo com o software instalado nele. Além disso, homogeneidade de plataforma e de dispositivos não é uma hipótese realista no ambiente de computação móvel atual [7, 25]. A diversidade de plataformas resulta em uma variedade de dispositivos, sistemas operacionais e recursos presentes em cada dispositivo.

Aplicações desenvolvidas no universo da computação ubíqua procuram estar sempre acessíveis aos seus usuários e essa diversidade acaba gerando diferentes versões finais da mesma aplicação, cada versão adaptada à plataforma na qual será executada. Se o dispositivo do usuário não possui o software necessário para apropriadamente acessar um arquivo, o usuário terá que iniciar algum processo de instalação desse software.

Aplicações Web podem ser uma opção para superar a necessidade de instalar software para manipular determinados dados apropriadamente. Aplicações Web são implementadas utilizando HTML5, CSS e JavaScript, e são distribuídas no formato de código fonte. Podem ser executadas em qualquer dispositivo que possua um navegador Web instalado, permitindo que um único código fonte seja executado em múltiplas plataformas.

No contexto de aplicações Web pode-se imaginar código e dados independentes de localização, ou seja, código e dados que não estão ligados a um local particular da Web. O código a ser executado e os dados a serem lidos/escritos podem ser armazenados tanto remota quanto localmente. Essa liberdade de localização é um dos principais atrativos das aplicações Web. Os usuários não estão mais restritos aos limites físicos de seus dispositivos, nem ao código que é executado e nem ao dado que esse código acessa.

Independência da localização do código da aplicação Web requer mover dinamicamente funcionalidades entre cliente e servidor, entre clientes na borda da rede e entre nós internos à rede. Além disso, o sistema que executa o código móvel deve se preocupar

com eficiência, interface com o usuário, segurança e alocação dos recursos disponíveis para iniciar a execução do código.

Mesmo com a independência de localização do código e dos dados acessados por aplicações Web, ainda existem informações locais no dispositivo do usuário que são importantes para manter a sessão de interação da aplicação. Um exemplo disso é o *token* [15] de acesso utilizado pelo lado cliente de uma aplicação para realizar requisições ao servidor. Outro exemplo são as configurações e customizações da interface gráfica com a qual o usuário interage. Essas informações não são enviadas para o servidor pois não afetam a forma na qual o conteúdo e código acessados pela aplicação do usuário devem ser armazenados e processados.

Existem esforços sobre o processo de coleta do estado da sessão de interação de aplicações Web, armazenamento do estado coletado para enviá-lo entre os dispositivos do usuário e recuperação do estado da sessão da aplicação no dispositivo alvo [7, 17, 21]. Essas abordagens apresentam algoritmos e técnicas que podem ser seguidos para realizar cada uma das ações referentes ao estado da sessão, avaliam como essas ações afetam o funcionamento da aplicação e sugerem estruturas para o armazenamento do estado da sessão.

Entretanto, essas abordagens não consideram como o mecanismo de transferência influencia no armazenamento do estado da sessão e no tempo total para a migração. Nessas abordagens é necessário que o usuário defina quais são os dispositivos fonte e alvo da migração, escolha qual o melhor dispositivo, entre os possíveis alvos, para continuar a sessão de interação da aplicação Web e interaja diretamente com os mecanismos de transferência. Ou seja, a escolha do dispositivo alvo da migração não é feita de forma automática e exige a interferência do usuário. Os eventos que podem dar início ao processo de migração também não são considerados por essas abordagens.

Coleta, armazenamento e recuperação são ações intermediárias no processo de migração da sessão de interação de aplicações Web e relacionam diretamente o estado da sessão com a aplicação Web executando em um dispositivo. Existem outras ações que afetam todo o processo, por exemplo, enviar efetivamente os dados do estado de sessão de interação de um dispositivo para outro ou iniciar o processo de migração. Há também informações sobre o usuário que podem auxiliar na escolha do dispositivo alvo da migração, por exemplo, as preferências com relação aos seus dispositivos, a agenda de compromissos, padrões de comportamento, localização física do usuário, etc.

Assim, há espaço para uma abordagem que dê transparência a todo o processo de migração do estado da sessão [20], levando em conta informações que podem ajudar na escolha do dispositivo alvo, eventos que podem iniciar o processo de migração e os mecanismos para efetivamente enviar o estado da sessão de interação entre os dispositivos do usuário. Não apenas transparência do processo de migração, mas também oferecer

uma camada para reduzir o esforço dos desenvolvedores para implementar ou modificar aplicações Web que necessitam da função de migrar suas sessões de interação com o usuário.

1.1.1 Preocupações com o processo de handoff

Entendemos *handoff* como toda a sequência de passos executada no processo para transferir a sessão de interação da aplicação de um dispositivo fonte para um dispositivo alvo. A qualquer momento pode ser necessário mudar o dispositivo com o qual o usuário está interagindo, por exemplo, devido a baixa carga de bateria, sobrecarga do processador, falta de espaço em memória, ausência dos periféricos adequados para interação com a aplicação, mudança da localização física do usuário, entre outros. Cada um desses exemplos pode gerar um evento que inicia o processo de *handoff* da sessão de interação da aplicação.

Uma vez determinado que a aplicação deve realizar o *handoff*, essa aplicação ou camada de suporte ao processo de *handoff*, por exemplo um *middleware*, deve, com o menor esforço possível por parte do usuário, migrar a sessão de interação da aplicação do dispositivo atual para um outro dispositivo. Nesse processo será necessário determinar o dispositivo alvo do *handoff*. É com esse dispositivo que o usuário dará continuidade a interação com a aplicação. Um exemplo de informações que podem ser utilizadas para a escolha do dispositivo alvo são as preferências do usuário em relação ao dispositivo alvo no qual deseja interagir com a aplicação ao final do *handoff*. Preferências como tamanho de tela, presença de determinado periférico, privacidade, tecnologias para comunicação (WiFi, Ethernet, 4G, Bluetooth, etc.), espaço em memória, capacidade de processamento, entre outras, podem ser utilizadas no processo de escolha do dispositivo alvo.

Durante o processo de *handoff*, também é necessário efetivamente enviar os dados da sessão de interação da aplicação que estão locais no dispositivo fonte para o dispositivo alvo. Esse envio pode ser feito por comunicação direta entre os dispositivos envolvidos, por um serviço intermediário ou por uma camada que dê suporte ao envio de dados. O envio pode ser gerenciado pela própria aplicação ou por um *middleware*.

1.2 Abordagem da solução

A abordagem proposta é baseada em um *middleware* por meio do qual aplicações Web podem adquirir a capacidade de migrar as sessões de interação com o usuário. O *middleware* foi nomeado "*Application Interaction Transfer middleware*" (AIT*m*). São consideradas as quatro principais ações do processo de *handoff*: os eventos que dão início ao *handoff*, a escolha do dispositivo alvo do *handoff*, a coleta, o armazenamento

e a restauração do estado da sessão de interação e os mecanismos de transferência do estado da sessão entre os dispositivos do usuário.

Apesar de considerar os eventos que podem dar início ao processo de *handoff*, este trabalho não tem como foco monitorar e notificar eventos. Consideramos que esses eventos são recebidos de componentes externos. Dessa forma, de acordo com a aplicação Web que fará uso da implementação do *middleware*, pode-se escolher livremente os eventos, dados relacionados a esses eventos e a forma na qual serão notificados.

Uma vez definido o alvo do *handoff*, o AIT*m* (*Application Interaction Transfer middleware*) sendo executado no dispositivo fonte recupera os dados locais da sessão de interação da aplicação. No contexto de aplicações Web são coletados os dados do DOM (Document Object Model), variáveis do arquivo JavaScript (JS File) e Cookies da aplicação.

Assim que esses dados chegam no dispositivo alvo, o AIT*m* no dispositivo alvo requisita o código da aplicação Web ao servidor que contém esse código e, após a aplicação Web estar carregada, realiza o processo para recuperar a sessão de interação. Dessa forma, o usuário pode voltar a interagir com a aplicação Web no mesmo estado da sessão que deixou no dispositivo fonte.

A fim de reduzir a quantidade de opções de implementação, estabelecer uma homogeneidade de plataforma e aplicações, este trabalho se concentra em aplicações Web. Este trabalho foca em cenários de interação sequencial, ou seja, quando um usuário executa uma aplicação em diferentes dispositivos e em diferentes momentos.

Como parte do trabalho, o AIT*m* foi implementado no formato de extensão de navegador que realiza o processo de *handoff* em favor das aplicações Web. A estrutura na qual o estado da sessão de interação é armazenado e a influência no tempo total do processo de *handoff* foram avaliados com respeito aos mecanismos para envio, através da rede, dos dados necessários para realizar o *handoff* da sessão de interação.

1.3 Contribuições

As abordagens encontradas na literatura apresentam algoritmos e técnicas que podem ser seguidos para realizar as ações de coleta, armazenamento e recuperação do estado da sessão de interação de uma aplicação Web. Essas abordagens também avaliam como essas ações afetam o funcionamento da aplicação e sugerem estruturas para o armazenamento do estado da sessão.

Entretanto, coleta, armazenamento e recuperação não são as únicas ações necessárias para completar todo o processo de *handoff*. Os eventos que iniciam o processo de *handoff*, a escolha do dispositivo alvo para dar continuidade a interação e como o meca-

nismo de transferência influencia no armazenamento do estado da sessão e no tempo total para a migração, também devem ser considerados.

A abordagem proposta neste trabalho é baseada em uma arquitetura de *middleware* que provê um grau de transparência e automaticidade ao processo de *handoff* do estado da sessão de interação de aplicações Web. Diferentemente de outras abordagens encontradas na literatura, as quatro principais funcionalidades do processo de *handoff* são tratadas em conjunto. A coleta e restauração do estado da sessão de interação, os eventos que dão início ao *handoff*, as informações que podem ajudar na escolha do dispositivo alvo e os mecanismos de transferência são realizados na forma de componentes da arquitetura.

1.4 Estrutura do trabalho

O restante deste documento está organizado como se segue. O Capítulo 2 apresenta conceitos necessários para o entendimento do trabalho. O Capítulo 3 apresenta a abordagem proposta. O Capítulo 4 apresenta uma implementação da abordagem proposta. O Capítulo 5 apresenta a avaliação. O Capítulo 6 discute trabalhos relacionados. Finalmente, o Capítulo 7 apresenta as conclusões e os trabalhos futuros.

CAPÍTULO 2

Fundamentaçao Teórica

Neste capítulo são apresentados os principais conceitos e técnicas que servem de base para o entendimento deste trabalho. São abordados: computação ubíqua, mecanismos para o envio de dados durante o processo de *handoff* e migração de sessão.

2.1 Computação Ubíqua

A Computação Ubíqua tem como base a onipresença da informática no cotidiano das pessoas. É um termo dado à terceira era da computação moderna [16], que é caracterizada pelo rápido crescimento na quantidade de dispositivos portáteis utilizados pelas pessoas, como *smartphones* e *tablets*. Tais dispositivos oferecem conexão à Internet ou a outros dispositivos, resultando em um mundo onde cada pessoa possui e usa vários computadores que podem interagir entre si.

O termo foi usado pela primeira vez por Mark Weiser (1952 – 1999) em 1988 no Xerox PARC, enquanto era diretor do Laboratório de Ciência da Computação (*Computer Science Laboratory* – CSL), e publicado em 1991 em seu artigo *The Computer for the 21st Century* [32]. A computação ubíqua tem como objetivo integrar a informática com as ações e comportamentos naturais das pessoas, de maneira que os usuários não percebem que estão dando comandos a um computador, mas atuam como se estivessem interagindo diretamente com os objetos do dia-a-dia.

A computação ubíqua apresenta pelo menos três princípios fundamentais: Diversidade, Descentralização e Conectividade [8]. Diversidade diz respeito aos dispositivos ubíquos, os quais trazem uma nova visão da funcionalidade em relação ao computador comum (PC), que possui propósito geral. Os dispositivos ubíquos visam um propósito específico, atendendo necessidades específicas de usuários particulares. Apesar de vários dispositivos poderem oferecer funcionalidades comuns, um pode ser mais apropriado para uma função do que outro. Por exemplo, um *smartphone* pode ser mais apropriado para fazer anotações rápidas, mas não é o melhor dispositivo para navegar na Web. Um *laptop* ou um *desktop* provavelmente será o dispositivo escolhido pelo usuário em sua casa para navegar pela Internet, com todo o potencial de recursos oferecidos pelo computador.

Um desafio trazido por essa diversidade refere-se ao gerenciamento das diferentes capacidades dos dispositivos, uma vez que cada um tem características e limitações próprias, tornando difícil oferecer aplicações comuns.

Sobre descentralização, na computação ubíqua as responsabilidades são distribuídas entre vários dispositivos, que assumem e executam certas tarefas e funções. Esses dispositivos cooperam entre si para a construção do contexto e da inteligência no ambiente, os quais são refletidos nas aplicações. Dessa forma, uma rede dinâmica de relações é necessária entre dispositivos e entre dispositivos e serviços do ambiente. Um ponto de destaque relacionado à distribuição inclui o gerenciamento, pelos serviços, das aplicações que executam nos dispositivos. Os serviços precisam manter registros de perfis de usuários e de dispositivos com capacidades diferentes. Além disso, o serviço deve ser flexível e robusto para tratar um vasto número de dispositivos clientes.

Sobre conectividade, espera-se uma comunicação sem fronteiras na qual dispositivos e aplicações movem-se juntamente com o usuário de forma transparente e entre diversas redes heterogêneas. A mudança de pontos de acesso, de tecnologia de comunicação ou de ambos não deve afetar o comportamento de uma aplicação ao ponto de gerar uma falha no serviço oferecido.

Uma outra abordagem seria o uso de computação sensível ao contexto. Essa tecnologia torna possível que dispositivos capturem seu contexto de execução automaticamente, sendo possível transferir e adaptar esse contexto ao novo ambiente de execução. Perera *et al.* [22] descrevem contexto como qualquer informação que pode ser utilizada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, local ou objeto considerado relevante na interação entre um usuário e uma aplicação, incluindo o próprio usuário e aplicações. Alguns exemplos são a presença de uma pessoa no espaço, tipos de movimento corporal (braços, dedos, cabeça, olhos, face) e dispositivos de saída disponíveis (monitores de televisão, computador ou *smartphones*), entre outros.

2.1.1 Aplicações Ubíquas

Roriz *et al.* [24] define aplicação ubíqua como um conjunto de ativações de uma aplicação, ou seja, é formada pelas instâncias da aplicação no ambiente de computação ubíqua. Por sua vez, uma aplicação é formada por seu código executável e seus metadados. Nos metadados estão o nome, versão e plataforma suportados pela aplicação. Cada instância local da aplicação é chamada de ativação da aplicação.

Aplicações ubíquas são desenvolvidas com base nos conceitos e tecnologias oferecidas pela computação ubíqua. Elas visam estar disponíveis aos usuários a todo momento e em qualquer lugar. Compartilham de três objetivos básicos: interação natural com as pessoas e com objetos do dia-a-dia, servindo de interfaces para ambientes com-

putacionais; uso de tecnologias inteligentes, sensíveis a diferentes contextos e atividades humanas e capazes de reagir a elas; e comunicação, tanto entre pessoas e objetos quanto entre objetos.

Aplicações ubíquas são integradas de forma natural às atividades diárias do usuário. Essas aplicações se adaptam ao dinamismo do ambiente do usuário a fim de estarem disponíveis a qualquer momento. A onipresença das aplicações ubíquas pode ser alcançada tanto pelo dispositivo que se move com o usuário quanto pela aplicação que se move entre dispositivos. Em ambos os casos, a aplicação precisará adaptar-se às mudanças tecnológicas e aos recursos oferecidos no ambiente [2].

Um exemplo dessas aplicações é a casa inteligente. Todo o ambiente da casa é controlado de forma harmônica, propiciando aos seus usuários controle dos gastos energéticos, controle de luminosidade dos ambientes de acordo com seu uso e controle dos principais equipamentos da casa, entre outros. Em outras palavras, cada vez menos será necessário que as pessoas selecionem as tarefas que os equipamentos deverão fazer porque os próprios equipamentos serão programados a partir da inferência das vontades e padrões de seus usuários.

2.2 Middleware

Normalmente, os usuários de aplicações as utilizam sem possuir o conhecimento dos detalhes técnicos, protocolos e gerenciamento das operações realizadas pelas aplicações. Os usuários não possuem esse conhecimento técnico e muitos não querem possuir, pois o seu interesse está na interação com as aplicações e não nas operações realizadas por elas a fim de atender aos comandos do usuário. Se a aplicação for distribuída, geralmente uma conexão deve ser feita, seguida de uma negociação para definir os parâmetros mais apropriados para a comunicação e execução das atividades da aplicação. Toda essa negociação e execução de atividades são escondidas da maioria dos usuários [6].

Middleware [4] é uma camada de software que provê um alto grau de abstração em programação distribuída entre a plataforma de execução, como o sistema operacional, e as aplicações. Utilizando um *middleware*, um programador pode desenvolver um software distribuído utilizando as mais apropriadas, corretas e eficientes soluções oferecidas pelo *middleware*. Ou seja, utilizar um *middleware* para construir um sistema distribuído livra os desenvolvedores de implementar detalhes de operações de baixo nível, por exemplo: controle de concorrência, gerência de transações e comunicação de rede. Assim, deixando os desenvolvedores livres para focar nos requerimentos da aplicação.

O *middleware* tem o papel de interligar diferentes aplicações em diferentes plataformas de execução em diferentes dispositivos. Ou seja, ele visa ocultar da melhor maneira possível a heterogeneidade das plataformas das aplicações. O *middleware* consiste

de um conjunto de serviços disponíveis que permite que múltiplos processos, executando em um ou mais dispositivos, interajam através de uma rede. Um dos principais objetivos do *middleware* é alcançar a transparência de distribuição [23], a capacidade que um sistema tem de ocultar entre os usuários e as diferentes aplicações de ser um sistema único de computador. Exemplos de transparências desejadas pelo *middleware* são: transparência de acesso, de localização, de migração, de relocação, de replicação, de concorrência e de falha.

De forma resumida, transparência de acesso oculta as diferenças na representação de dados e no modo como os recursos podem ser acessados pelas aplicações dos usuários. Transparência de localização oculta o lugar em que um recurso está localizado. Fazendo uso da transparência de localização, a transparência de migração oculta que um recurso pode ser movido para outra localização e a transparência de relocação oculta que um recurso pode ser movido para outra localização enquanto em uso. Transparência de replicação oculta o fato de um recurso ter várias cópias. Transparência de concorrência oculta que o mesmo recurso está sendo usado por várias aplicações ao mesmo tempo. Transparência de falha oculta a falha e a recuperação de um recurso sem a percepção do usuário.

2.3 Migração de sessão

Migração ou mobilidade de sessão acontece quando uma sessão estabelecida com um ou mais serviços precisa ser movida para outro dispositivo no qual a execução irá continuar. O usuário dever ser capaz de continuar a utilizar a aplicação no mesmo ponto no qual parou. Mobilidade de sessão também poder ser referente ao serviço, uma vez que ele seja movido de um dispositivo/terminal para outro.

2.3.1 *Liquid Software*

Para criar um software que se adéque ao paradigma de multi-dispositivos, a arquitetura atual das aplicações Web precisa ser redesenhada para habilitar o que é conhecido na literatura como *Liquid Software* [12]. *Liquid Software* é toda a infraestrutura para dinamicamente mover funcionalidades e aplicações entre cliente e servidor, entre clientes e entre nós internos da rede. Tais aplicações não apenas aproveitam totalmente as vantagens dos recursos de computação, armazenamento e comunicação dos dispositivos do usuário final, mas também podem facilmente e dinamicamente migrar de um dispositivo para outro seguindo o usuário.

É uma abordagem na qual aplicações e dados podem fluir de um dispositivo, ou tela, para outro sem interrupções, permitindo que o usuário mude de dispositivos sem

se preocupar com o gerenciamento desse *handoff*. Em um ambiente com suporte completo a *Liquid Software* os dispositivos gerenciam operações como: *backup*; instalação, atualização e reinicialização de aplicações; migração de contas; cópia das preferências e configurações do usuário; etc.

Liquid Software pode ser dividido nas seguintes categorias [27]:

- Interação Sequencial (*Sequential Screening*): um único usuário interage com uma aplicação em diferentes dispositivos em diferentes momentos. A aplicação se adapta à capacidade do dispositivo atual, mas mantém atendendo as necessidades do usuário no novo contexto.
- Interação Simultânea (*Simultaneous Screening*): um único usuário interage com diversos dispositivos ao mesmo tempo, ou seja, a sessão da aplicação está aberta e executando em mais de um dispositivo ao mesmo tempo. Diferentes dispositivos interagem com o usuário de acordo com suas capacidades, realizando adaptações quando necessário.
- Cenário colaborativo (*Collaboration Scenario*): vários usuários executam a mesma aplicação em seus dispositivos. Essa colaboração pode ser sequencial ou simultânea.

Todas as categorias acima compartilham os mesmos desafios para adaptar a interface do usuário e sincronizar dados e estados entre os dispositivos.

Os seguintes itens [19] resumem os principais requerimentos para um *Liquid Software*:

1. Os usuários devem ser capazes de mudar o dispositivo com o qual estão interagindo a qualquer momento, de forma fácil e sem interferir na função oferecida pela aplicação.
2. Todos os aspectos referentes à gerência e manutenção dos dispositivos devem ser minimizados ou completamente removidos das operações realizadas pelo usuário.
3. Dados e sessões das aplicações do usuário devem ser transferidos ou sincronizados transparentemente entre os dispositivos.
4. A migração do usuário não deve ser limitada pela heterogeneidade de hardware e software, mas possível entre qualquer dispositivo que possua a capacidade de executar a aplicação que está sendo utilizada pelo usuário.
5. O usuário deve ser capaz de determinar as funções e dados que devem mudar de dispositivo. Se o usuário decidir que determinada função deve permanecer em determinado dispositivo, tal expressão de preferência deve ser feita da forma mais intuitiva possível.

2.3.2 Comportamento *Follow-me*

Aplicações com a característica *follow-me* [28] devem seguir o usuário de acordo com seu movimento. Esse movimento pode ser de componentes lógicos (dados, código e processamento) e/ou de componentes físicos (periféricos e dispositivos em uso) [1].

O usuário agora pode desfrutar de, no mínimo, duas formas de mobilidade:

1. O usuário transfere a aplicação para qualquer dispositivo viável, fazendo um melhor uso dos recursos disponibilizados por esse dispositivo.
2. A aplicação pode tomar a decisão de migrar para algum dispositivo, seguindo seu usuário a fim de sempre estar disponível para uso.

Assim, a aplicação migra para o dispositivo no qual seja possível, mais conveniente ou mais eficiente a interação com o usuário. Dessa forma, a experiência do usuário não é interrompida, ou é interrompida pela menor quantidade de tempo possível.

2.4 Mecanismos para envio de dados

Um dos passos durante o processo de *handoff* é enviar os dados do estado local de um dispositivo fonte para um dispositivo alvo. Assim, o usuário pode continuar a interagir com a mesma aplicação, mantendo a mesma sessão de interação mas em um dispositivo diferente. Esta seção apresenta alguns possíveis mecanismos que podem ser utilizados para o envio de dados durante o processo de *handoff*.

2.4.1 Serviço em Nuvem

De acordo com o NIST (*The National Institute of Standards and Technology*), Computação em Nuvem (*Cloud computing*) é um modelo que possibilita acesso a um conjunto de recursos computacionais compartilhados e configuráveis (ou seja, rede, servidores, armazenamento, aplicações e serviços) de modo ubíquo, conveniente, sob demanda, de forma rápida e com mínimo esforço de gerenciamento ou interação com o provedor de serviço [18].

Computação em nuvem refere-se à utilização da memória e da capacidade de armazenamento e processamento de computadores e servidores compartilhados e interligados por meio da Internet, seguindo o princípio da computação em grade [11]. O armazenamento de dados é feito em serviços que podem ser acessados de qualquer lugar do mundo, a qualquer hora, não havendo necessidade de instalação de programas ou de armazenar dados localmente. O acesso a programas, serviços e arquivos é remoto, através da Internet, sendo esse o motivo da alusão à nuvem.

Há três tipos fundamentais de modelos de serviços na computação em nuvem [18]: *Infrastructure as a Service* – IaaS, *Plataform as a Service* – PaaS e *Software as a Service* – SaaS. IaaS é o fornecimento de hardware (servidor, armazenamento e rede), e software (tecnologias de virtualização de sistemas operacionais, sistema de arquivo), como um serviço. Não requer um contrato de longo termo de uso e permite ao usuário requirir recursos em demanda [5].

PaaS é a ideia de prover hardware e software, assim como em IaaS, mais um conjunto de ferramentas (aplicações, banco de dados, funções de programação), sobre as quais o usuário pode construir suas próprias aplicações. PaaS é uma plataforma oferecida através da Web para desenvolvimento e implementação de aplicações para os desenvolvedores.

SaaS é a ideia de oferecer um conjunto de ferramentas e aplicações (executando sobre uma plataforma e infraestrutura) que o usuário irá pagar a medida em que for utilizando os recursos. O usuário não precisa desenvolver ou programar os recursos oferecidos, mas é necessário configurá-los para atender as necessidades que o fizeram contratar o serviço. Um provedor SaaS geralmente fornece e gerencia uma aplicação em seu próprio *data center* e torna a aplicação disponível para múltiplos usuários através da Web. Alguns provedores SaaS utilizam outros provedores PaaS ou IaaS para fornecer seus serviços [3, 5].

Estabelecendo uma conexão com um serviço em nuvem, pode-se utilizar as ferramentas oferecidas por ele, podendo ser desde um editor de textos, um jogo ou um editor de vídeos, até um serviço de localização ou de processamento de padrões em imagens. Enquanto os servidores lidam com o processamento intensivo ou são responsáveis pela maioria ou pelo completo armazenamento dos dados, o dispositivo cliente fica responsável por interagir com o usuário, enviando as informações inseridas pelo usuário para o serviço em nuvem e, ao receber a resposta bem sucedida ou não da requisição feita ao serviço, apresentar os dados recebidos de forma que o usuário compreenda o que está acontecendo.

Uma das vantagens dos serviços em nuvem, para a maioria dos usuários, é a não necessidade de ter uma máquina com todos os recursos necessários para a execução de uma aplicação, uma vez que os processos que consomem mais recursos ou que necessitam de um software ou hardware específicos são executados em servidores remotos. A elasticidade na alocação de recursos torna-se um atrativo para aplicações que visam aumentar o número de usuários atendidos, assim como fornecer novos serviços para usuários já alcançados. No entanto, o armazenamento em nuvem também gera desconfiança, principalmente no que se refere à segurança. A proposta é manter informações importantes e serviços essenciais em um ambiente virtual, e não são todos os usuários que se sentem à vontade com isso [33].

Outro fator de destaque é a primordial necessidade de conexão com a Internet, pois os serviços estão localizados remotamente. Essa conexão deve ser estável e rápida, principalmente quando se trata de *streaming* de dados, pois a informação está sendo consumida pelo usuário enquanto os dados estão sendo enviados. Deve-se levar em conta que os servidores ficam em lugares distantes. Portanto, uma conexão à Internet instável ou de baixa velocidade é prejudicial para o aproveitamento pleno da tecnologia.

2.4.2 Protocolo de comunicação Gossip

Um protocolo de comunicação *Gossip* (fofoca em Português), também chamado de protocolo epidêmico (*epidemic protocol*), é um protocolo de comunicação entre máquinas inspirado na forma como fofocas se espalham no meio social ou na forma como um vírus se espalha em um ambiente biológico [13]. Um protocolo de comunicação *Gossip* pode ser implementado por aplicações que precisam disseminar, agregar ou sincronizar informações, balancear carga ou gerenciar redes. A propriedade comum dessas implementações refere-se ao fato de que, periodicamente, cada nó da rede troca informações com um de seus pares. Os nós que trocam informações são escolhidos a partir de uma lista de pares. Tal lista é fundamental para o funcionamento do protocolo *Gossip*.

De forma resumida, um protocolo *Gossip* funciona da seguinte forma:

1. Um nó da rede escolhe aleatoriamente um par de sua lista de pares, também chamada de visão (*view*), para trocar dados.
2. A troca de dados ocorre entre esses nós.
3. Cada nó processa os dados recebidos.
4. A lista de pares de cada nó é atualizada de acordo com a implementação de *Gossip* escolhida.
5. Esses passos são repetidos periodicamente por cada nó da rede a fim de disseminar os dados.

Uma implementação do protocolo *Gossip* é o CYCLON [31]. O CYCLON é implementado sobre sistemas P2P não estruturados. Esse sistema visa explorar a aleatoriedade para disseminar informação através de um grande conjunto de nós da rede. Um ponto de foco do CYCLON está em manter a camada (*overlay*) do conjunto de nós conectada mesmo após algum desastre ocorrer sem manter qualquer informação global ou exigir qualquer intervenção administrativa.

2.5 Considerações

Como apresentado na seção 2.1, a computação ubíqua tem como base a onipresença da informática no cotidiano das pessoas. Migrar a sessão de interação da aplicação entre os dispositivos do usuário é uma das abordagens para alcançar essa onipresença. O *handoff* da sessão deve ser feito de forma a seguir o movimento do usuário, tanto de componentes lógicos quanto de componentes físicos, levando em conta as restrições de cada dispositivo.

A seção 2.3.1 apresenta do conceito de *Liquid Software* e três categorias: interação sequencial, interação simultânea e cenário colaborativo. Este trabalho foca-se nas aplicações de interação sequencial, realizando o *handoff* da sessão de interação das aplicações Web de um único usuário entre seus dispositivos que são utilizados em diferentes momentos.

Durante o processo de *handoff* é necessário efetivamente enviar os dados da sessão de interação da aplicação de um dispositivo para outro. Esse envio pode ser feito diretamente entre os dispositivos ou através de um intermediário. Caso se opte por um intermediário, há opções como um serviço em nuvem ou uma camada de software que implementa um protocolo de comunicação.

CAPÍTULO 3

Arquitetura

Este capítulo discorre sobre os aspectos arquiteturais da abordagem proposta a fim de descrever os conceitos que representam a contribuição do trabalho.

3.1 Escopo do trabalho

Dentro da diversidade de aplicações ubíquas, a abordagem desenvolvida neste trabalho tem como foco aplicações Web interativas, em particular aplicações com as quais os usuários interagem constantemente por meio de uma interface gráfica. Exemplos de aplicações consideradas neste trabalho são editores de texto e de imagens, reprodutores de áudio e vídeo e jogos. Ou seja, não serão estudadas aplicações que desempenham suas funções em plano de fundo, como, aplicações de monitoramento de pessoas e locais e aplicações que fornecem serviços para outras aplicações.

Aplicações nativas (*stand alone*) para computador ou *smartphone*, ou seja, aplicações instaladas diretamente nos dispositivos possuem uma versão final equivalente para cada tipo de plataforma nas quais podem ser executadas. É necessário desenhar uma camada *middleware* para dar suporte à migração da sessão da aplicação de um dispositivo para outro de forma que a aplicação fique livre para desempenhar apenas a função para a qual foi desenvolvida.

Aplicações nativas não são portáveis em virtude das diferenças entre as plataformas, para seguir essa abordagem, para cada plataforma é necessário manter implementações distintas das aplicações, assim como versões específicas da camada de *middleware*. Entretanto, este trabalho não pretende manter o controle de tantas versões. Assim, a pesquisa tem como foco um único tipo de plataforma de execução de aplicações, navegadores Web. O uso de uma plataforma padronizada (navegadores Web e ferramentas associadas) garante a portabilidade.

3.2 Arquitetura da abordagem proposta

A arquitetura da abordagem pode ser vista na Figura 3.1. Na arquitetura se destacam: o Navegador Web (*Web Browser*), Eventos (*Event*), o Método de Transferência (*Transfer Method*), o *Middleware* e a Aplicação Web (*Web Application*).

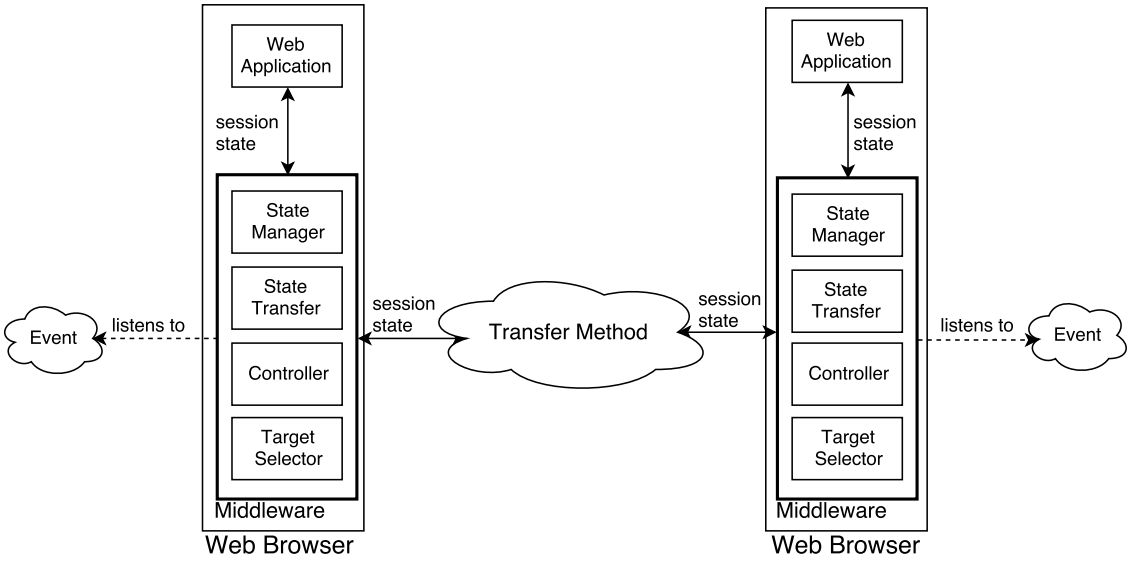

Figura 3.1: *Arquitetura do abordagem.*

Em geral, o processo de *handoff* envolve um dispositivo fonte e alvo, cada um executando um navegador Web, representado por *Web Browser*. No navegador Web são executadas aplicações Web que podem ter seu estado de sessão de interação com o usuário migrado entre os navegadores Web nos dispositivos do usuário. Uma aplicação Web é representada por *Web Application*.

Como apresentado na Seção 1.2, são consideradas as quatro principais ações do processo de *handoff*: eventos que dão início ao *handoff*, a escolha do dispositivo alvo, a coleta e restauração do estado da sessão e o mecanismo de transferência. Na arquitetura da abordagem, essas quatro ações estão representadas respectivamente por *Event*, *Target Selector*, *State Manager* e *State Transfer*. A tecnologia de transferência intermediária entre as instâncias do *middleware* nos navegadores Web é representado por *Transfer Method*.

Externos ao *middleware* estão os eventos. Esses eventos podem estar relacionados ao navegador Web, ao dispositivo do usuário, a aplicação, ou a algum estímulo externo. Internamente, o *middleware* escolhe o dispositivo alvo do *handoff*, captura o estado de sessão de interação das aplicações e utiliza o mecanismo de transferência para enviar esse estado para outra instância do *middleware*.

O *Middleware* é o componente central da arquitetura proposta, chamado AIT*m* neste trabalho. Ele opera como uma camada de *software* no navegador e é responsável

pelo gerenciamento e execução das ações necessárias para o processo de *handoff* da sessão de interação da aplicação Web. Esse gerenciamento é feito pelo componente *Controller*. É responsabilidade do AIT*m* obter o estado da sessão de interação da aplicação no dispositivo fonte e restaurá-lo no dispositivo alvo para dar continuidade a interação do usuário com a aplicação. A Figura 3.2 ilustra com mais detalhes a arquitetura do AIT*m* e como seus componentes se relacionam.

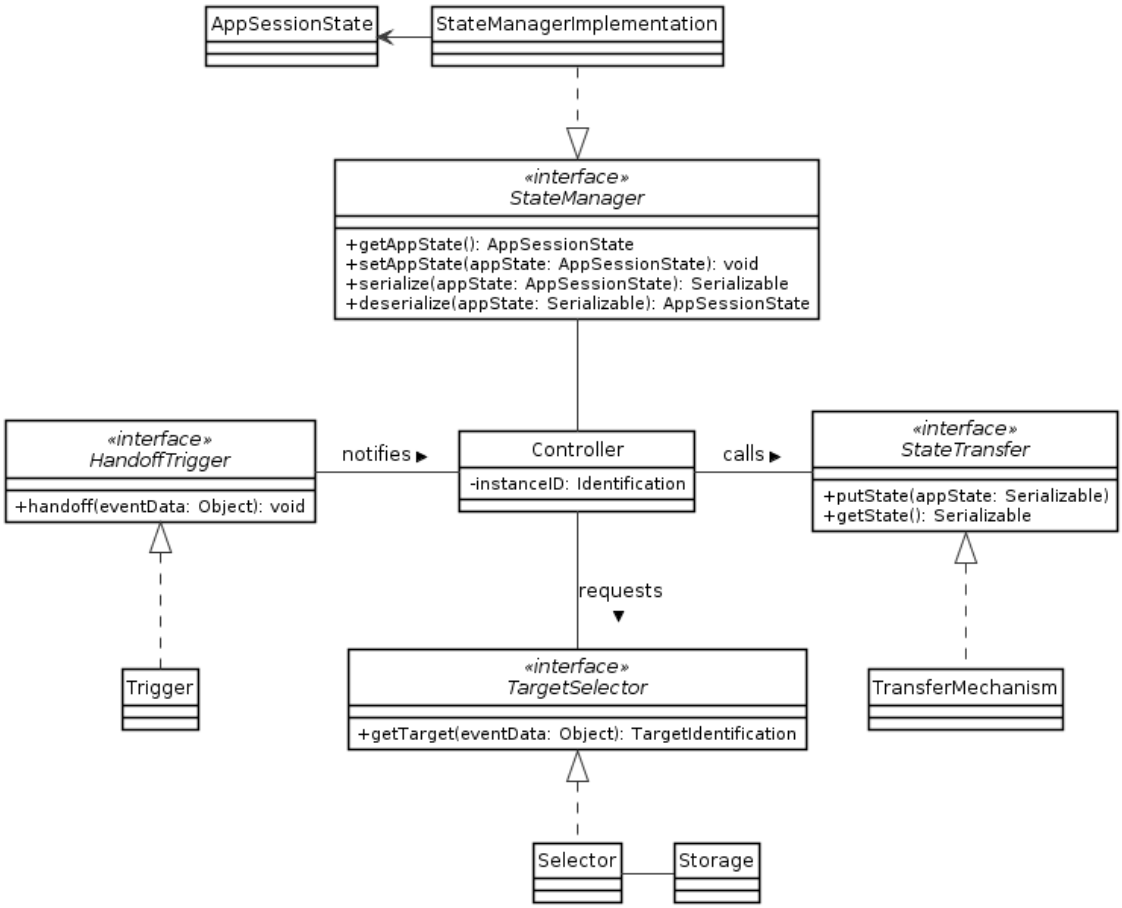

Figura 3.2: *Diagrama de classes do AITm.*

Este trabalho requer que o navegador Web ofereça uma forma para instalar ou adicionar o AIT*m*, sem o qual não seria possível a interação entre o AIT*m* e as aplicações Web. É necessário um mecanismo que permita a comunicação entre as instâncias do AIT*m* em diferentes dispositivos, por exemplo, uma rede sem fio como o *Wi Fi* (IEEE 802.11), *Bluetooth* (IEEE 802.15), uma rede cabeada *Ethernet* (IEEE 802.3), WebSockets [10], *Web Real-Time Communications* (WebRTC) [14], etc. A implementação e escolha da tecnologia do mecanismo de transferência é ortogonal a arquitetura proposta, não interferindo no funcionamento do *middleware*.

3.2.1 Controle do AIT*m* e captura do estado da sessão de interação

No centro da arquitetura do AIT*m* está o componente *Controller*. Esse componente é responsável por controlar e monitorar a execução das tarefas dos outros componentes do AIT*m*. Após receber a notificação de um evento que inicia o processo de *handoff*, o *Controller* requisita a implementação do *TargetSelector* o dispositivo alvo do *handoff*. Uma vez decidido o alvo, o *Controller* requisita o estado da sessão de interação da aplicação à implementação do *StateManager* e depois encaminha esse estado para o mecanismo de transferência que implementa o *StateTransfer*.

No navegador Web estão executando uma ou mais aplicações, cada aplicação com seu próprio estado e dados de sua sessão de interação. Durante o processo de *handoff*, obter e restaurar o estado da sessão de interação da aplicação é uma tarefa do componente *StateManagerImplementation* que implementa a interface *StateManager*. Na abordagem proposta, existem dois métodos para o *StateManagerImplementation* obter ou restaurar o estado da sessão de interação: em nível de aplicação, trocando mensagens com a aplicação, ou em nível de sistema, trocando mensagens com o navegador Web. Pode-se escolher entre os métodos adaptando-se a implementação do componente *StateManagerImplementation*.

Em nível de aplicação, uma forma de permitir que o *StateManagerImplementation* troque mensagens com a aplicação é oferecendo uma *Application Programming Interface* (API) ou pela instrumentação do código fonte da aplicação. Essa abordagem exige modificação do código fonte da aplicação Web para adicionar a API ou para realizar a instrumentação, permitindo que o *StateManagerImplementation* comunique-se com a aplicação durante o processo de *handoff*. O *StateManagerImplementation* utiliza a API para enviar mensagens a aplicação a fim de obter ou restaurar o estado da sessão de interação.

Em nível de sistema, as mensagens para obter e restaurar a estado da sessão de interação da aplicação são enviadas do *StateManagerImplementation* para o navegador Web. Interagir diretamente com o navegador livra a aplicação de ter seu código fonte modificado. A troca de mensagens entre o *StateManagerImplementation* e o navegador Web é feita através dos métodos fornecidos pelo navegador. Assim, é possível o *StateManagerImplementation* requisitar os dados para obter ou restaurar o estado da sessão de interação da aplicação trocando mensagens com a aplicação ou com o navegador.

No AIT*m*, o estado de sessão de interação de uma aplicação é representado pelo componente *AppSessionState*. Os dados contidos no estado da sessão de interação variam de acordo a função que a aplicação oferece. Por exemplo, uma aplicação de edição de texto pode ter um estado de sessão de interação diferente de uma aplicação de reprodução de vídeo. Independentemente das funções que cada aplicação oferece, a maioria das aplicações Web são implementadas utilizando HTML (*HyperText Markup*

Language), CSS (*Cascading Style Sheets*) e JavaScript, entretanto, caso alguma aplicação utilize outras tecnologias, essa exceção pode ser tratada pela implementação da interface *StateManager*.

Existem casos especiais nas aplicações Web que devem ser tratados durante a captura e restauração do estado de sessão. Exemplos são: temporizadores (*timers*), *function closures*, objetos de mídia HTML5, conexões de transmissão de dados. Esses casos especiais não são tratados neste trabalho pois já foram cobertos por trabalhos como Lo *et al.* [17] e Oh *et al.* [21]. Assim, o AIT*m* espera por um estado de quiescência para realizar a captura ou restauração do estado de sessão. Um estado de quiescência é aquele no qual não há operações pendentes a serem feitas, por exemplo, o *download* não finalizado de um arquivo. Entretanto, o AIT*m* pode ser adaptado para dar suporte a esses casos especiais.

3.2.2 Disparo do processo de *handoff*

Baixa carga de bateria restante no dispositivo, disponibilidade de outro dispositivo que ofereça periféricos de entrada ou saída mais apropriados, baixo desempenho da rede na qual o dispositivo está conectado, mudança do local físico do usuário (por exemplo, sair do quarto e ir para a cozinha), um botão na interface da aplicação, um sensor de presença em uma sala, uma etiqueta NFC (*Near Field Communication*) com a qual o dispositivo entra em contato, a predição do local físico do usuário, a análise do padrão de interação do usuário com seus dispositivos, entre outros, são exemplos de eventos que podem dar início ao processo de *handoff*. Os eventos podem estar relacionados ao navegador Web, ao dispositivo do usuário, a aplicação ou a algum estímulo externo.

O componente *Trigger* representa cada possível implementação de um evento. A interface *HandoffTrigger* é implementada pelo componente *Trigger* e é utilizada para notificar o *Controller* que o processo de *handoff* deve ser iniciado. Desse modo, a implementação do *Trigger* não fica restrita a um único evento, podendo ser adaptada de acordo com o evento escolhido para dar início ao *handoff*. Os eventos e tecnologias utilizados para implementar esses eventos são ortogonais ao *middleware*, não mudando a forma como as funções do AIT*m* são desempenhadas. Assim, o principal propósito da implementação do componente *Trigger* é notificar os dispositivos do usuário quando um processo de *handoff* deve ser iniciado e fornecer, se necessário, os dados relacionados a esse evento.

3.2.3 Mecanismo de Transferência

O mecanismo de transferência é responsável pela comunicação entre as instâncias do AIT*m* durante o processo de *handoff*. Essa comunicação tem o propósito de en-

viar os dados da sessão de interação das aplicações. Essa comunicação é gerenciada pelo componente *Controller*, sendo que o mecanismo de transferência é dedicado apenas a migração da sessão e não é utilizado diretamente pelas aplicações. O AIT*m* é independente da tecnologia utilizada pelo mecanismo de transferência. A interface *StateTransfer* é utilizada pelo *Controller* para interagir com o mecanismo de transferência que implementa essa interface. Cada possível mecanismo de transferência é representado pelo componente *TranferMechanism*.

Por exemplo, uma implementação do *TranferMechanism* pode utilizar o paradigma cliente–servidor, sendo o cliente o *middleware* e o servidor um serviço hospedado em um ambiente de nuvem. Esse serviço é utilizado como intermediário para o envio dos dados entre os dispositivos fonte e alvo do *handoff*. Essa abordagem permite que as instâncias do AIT*m* em dispositivos que estão em diferentes redes privadas troquem dados de sessão de interação e, como extra, permite que os dados da sessão da aplicação sejam salvos em um sistema de armazenamento. Isto contribui para a resiliência da abordagem, podendo ser utilizado para restaurar a sessão da aplicação caso algum erro ocorra nos passos executados durante o processo de *handoff*.

Caso os dispositivos envolvidos na transferência estejam na mesma rede privada, uma abordagem que utilize comunicação direta entre eles pode ser vantajosa. Com a comunicação direta, uma conexão é criada entre os dispositivos, com poucos saltos entre eles, podendo assim reduzir o tempo gasto para o envio dos dados. Entretanto, esta abordagem não proporciona a resiliência que a anterior oferece.

Também é possível utilizar um protocolo de comunicação que sincronize periodicamente os dados da sessão de interação da aplicação entre os dispositivos do usuário, tal como um protocolo de *gossiping*. Com essa abordagem, o estado da sessão de interação é sincronizado periodicamente entre os dispositivos do usuário. Assim, durante o processo de *handoff* é necessário apenas transferir o controle da aplicação entre os dispositivos. Além de oferecer uma forma de resiliência, pois o estado da sessão é replicado entre os dispositivos.

3.2.4 Escolha do alvo do processo de *handoff*

Na arquitetura proposta, o dispositivo fonte do processo de *handoff* é o que está sendo utilizado pelo usuário no momento que o evento de início do *handoff* acontece, assim, será necessário decidir apenas o dispositivo alvo. O *Controller* requisita ao componente *Selector* o dispositivo alvo do *handoff* através da interface *TargetSelector*. Dependendo da implementação do *Selector*, é possível que esse componente precise de informações sobre o evento que iniciou o *handoff* ou de outras informações previamente inseridas para decidir qual o dispositivo alvo. Informações sobre o evento podem ser for-

necidas pelo *Controller* e outras informações podem ser previamente inseridas e armazenadas em um sistema de armazenamento. Esse sistema de armazenamento é representado pelo componente *Storage*.

Independentemente do *Selector* utilizar informações previamente inseridas ou as descobrir em tempo de execução, a resposta ao *Controller* será um identificador do dispositivo alvo. O identificador do dispositivo pode ser um número, uma *string*, um endereço IP ou qualquer outra forma que identifique de maneira única um dispositivo. Com esse identificador, o *Controller* do AIT*m* no dispositivo fonte pode conseguir informações para trocar mensagens com o *Controller* do AIT*m* no dispositivo alvo. Por exemplo, se as instâncias do AIT*m* se comunicam diretamente, uma informação necessária será o endereço IP do dispositivo alvo. Com esse endereço IP, o mecanismo de transferência pode enviar os dados do estado da sessão de interação do dispositivo fonte para o alvo e o *Controller* no dispositivo fonte poderá notificar o alvo que a interação do usuário com a aplicação continuará nesse novo dispositivo.

Caso seja utilizado um intermediário na comunicação entre as instâncias do AIT*m*, as informações necessárias dependerão de como esse intermediário funciona. Por exemplo, se o intermediário for um serviço de *publish/subscribe*, uma informação importante são os tópicos nos quais os dispositivos alvo estão inscritos. Assim, o *Controller* do AIT*m* no dispositivo fonte publica no tópico que o *Controller* do AIT*m* no dispositivo alvo está inscrito, notificando o alvo que a interação do usuário com a aplicação continuará nesse novo dispositivo.

3.2.5 Sequência de passos do *handoff*

O processo de *handoff* de uma sessão de interação de aplicação pode ser representado na forma de um diagrama de sequência. A Figura 3.3 apresenta a sequência de passos realizados pelos componentes do AIT*m* no dispositivo fonte do *handoff*. A Figura 3.4 apresenta a sequência de passos realizados pelos componentes do AIT*m* no dispositivo alvo do *handoff*. A ordem das mensagens foi enumerada para facilitar a explicação.

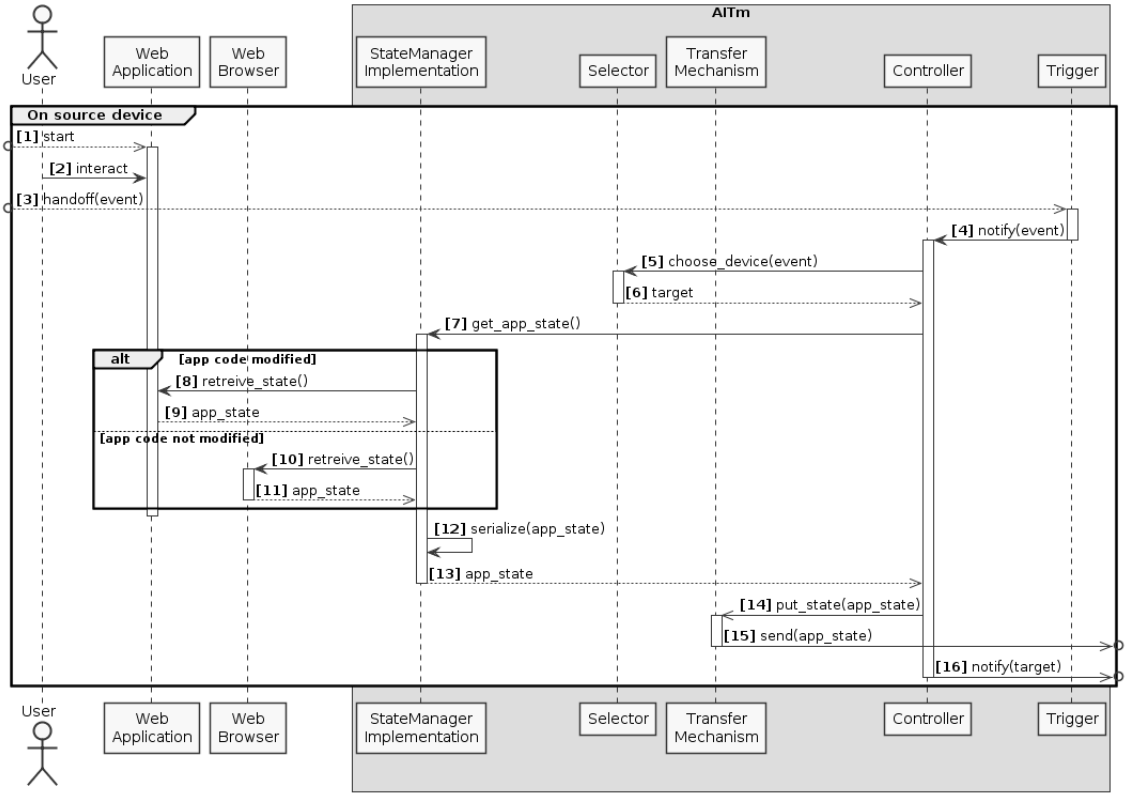

Figura 3.3: *Diagrama de sequência para o dispositivo fonte do handoff.*

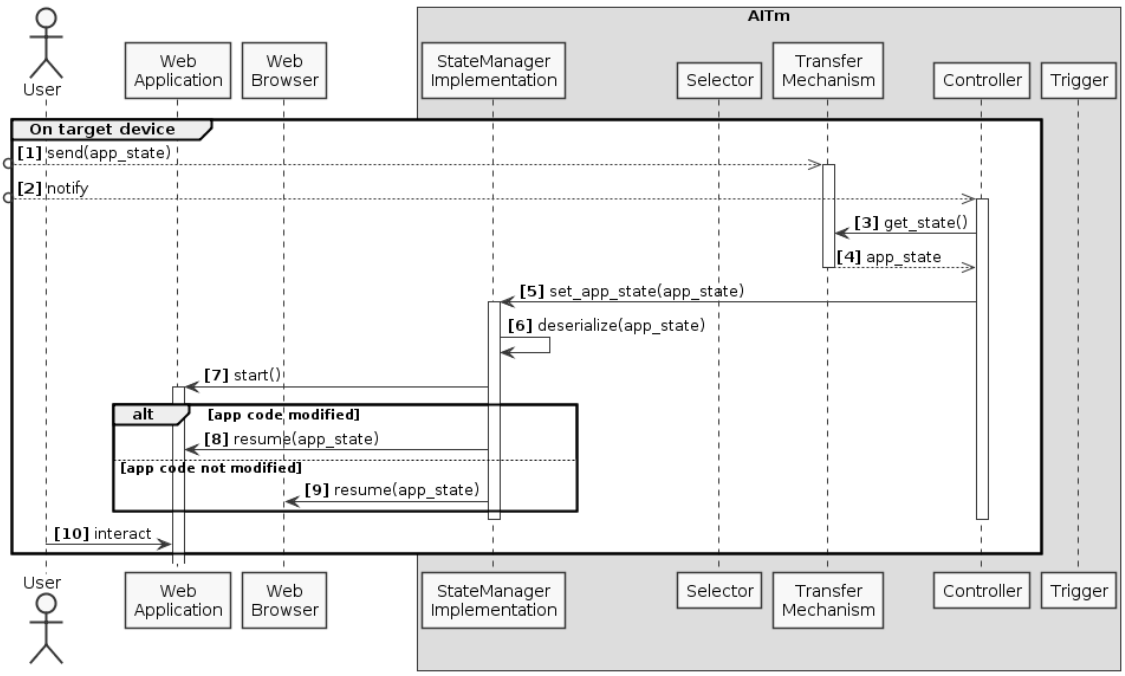

Figura 3.4: *Diagrama de sequência para o dispositivo alvo do handoff.*

Na Figura 3.3, no dispositivo fonte do *handoff*, em (1) a aplicação Web é iniciada pelo usuário ou por um processo, por exemplo, o AIT*m*. Em (2), o usuário interage com

a aplicação Web em execução, utilizando as funcionalidades oferecidas por ela. Em (3), um evento que determina o início do processo de *handoff* acontece e uma mensagem contendo dados relacionados ao evento é enviada para a instância do AIT*m* no dispositivo fonte.

Os dados dessa mensagem dependem dos eventos escolhidos para dar início ao processo de *handoff* e, consequentemente, da implementação que o componente *Trigger* oferece para a interface *HandoffTrigger*. Por exemplo, se o evento escolhido for a mudança do local físico do usuário, os dados da mensagem poder ser sobre esse novo local. Outro exemplo de evento é a probabilidade do usuário estar em um determinado local e utilizar um determinado dispositivo, nesse exemplo, os dados da mensagem podem ser o dispositivo, o local e o momento em que o usuário irá utilizá-lo.

Em (4), o componente *Trigger* notifica o *Controller*, componente que irá coordenar os passos que devem ser seguidos para realizar o *handoff*. Em (5) e (6), é escolhido o dispositivo alvo do *handoff* de acordo com a implementação do componente *Selector*. Em (7), é iniciado o processo para obter o estado da sessão de interação da aplicação Web. Caso o método para obter o estado de sessão de interação seja em nível de aplicação, então a aplicação teve seu código fonte modificado para integrar a API fornecida pelo AIT*m*. Assim, em (8) e (9) ocorre a troca de mensagens entre o *StateManagerImplementation*, que implementa a interface *StateManager*, e a aplicação Web.

Caso o método escolhido seja em nível de sistema, então não é necessário modificar o código fonte da aplicação Web e seu estado de sessão é obtido pela troca de mensagens entre o *StateManagerImplementation* e o navegador Web (10 e 11). Uma vez obtidos os dados de estado da sessão, eles são serializados para o envio do dispositivo fonte para o dispositivo alvo (12 e 13). Em (14), o *Controller* encaminha o estado da sessão de interação para o componente *TransferMechanism*, responsável por enviar o estado da sessão pela rede. Em (15), o envio dos dados da sessão é efetivamente realizado e em (16) a instância do AIT*m* no dispositivo fonte notifica a instância do AIT*m* no dispositivo alvo, informando que os dados foram enviados e que agora o dispositivo alvo deve assumir a execução da aplicação Web.

Na Figura 3.4, no dispositivo alvo do *handoff*, em (1), os dados do estado da sessão de interação da aplicação são recebidos. Em (2), a instância do AIT*m* no dispositivo alvo do *handoff* recebe a notificação para assumir a execução da aplicação Web. Em (3) e (4), o *Controller* recupera os dados do estado da sessão recebidos pelo *TransferMechanism*. Em (5) e (6), os dados são deserializados para que a aplicação Web possa retomar sua execução no dispositivo alvo. Em (7), a aplicação Web é iniciada pelo AIT*m* caso não esteja executando no dispositivo alvo. Em (8), o estado da sessão de interação é efetivamente inserido na aplicação Web no dispositivo alvo. O passo (8) ocorre caso o método escolhido seja em nível de aplicação. Caso o método escolhido seja

em nível de sistema, então o passo (9) é realizado, o qual também insere o estado da sessão de interação da aplicação, entretanto trocando mensagens com o navegador Web. Finalmente, em (10), o usuário volta a interagir com a aplicação Web, que agora está executando no dispositivo alvo do *handoff*.

3.3 Considerações

Neste capítulo foram apresentados os aspectos arquiteturais da abordagem proposta que descrevem os conceitos que representam a contribuição do trabalho. O processo de *handoff* do estado de sessão de interação de aplicações conta com quatro ações principais. Na abordagem proposta, cada uma dessas ações é abstraída na forma de interfaces, que são concretizadas por implementações. Assim, a abordagem pode ser adaptada de acordo com o conjunto de eventos, de especificidades das aplicações Web, de métodos de escolha do dispositivos alvo e de mecanismos de transferência escolhidos.

Por exemplo, se uma das especificidades da aplicação Web é utilizar um método para autenticação de usuário, o fluxo de autenticação desse método pode ser automatizado pela implementação do componente *StateManagerImplementation*, permitindo que o usuário permaneça autenticado na aplicação Web mesmo após o *handoff*. Dessa forma, mesmo que o dispositivo alvo seja de uma plataforma de execução diferente, esteja conectado em uma rede diferente, com um endereço IP diferente ou qualquer outra característica diferente que seja verificada pelo método de autenticação, o usuário continuará autenticado ou poderá ser reautenticado.

Outro exemplo pode ser relacionado ao mecanismo de transferência. Cada mecanismo segue um padrão ou protocolo no qual recebe e responde requisições. Assim, a implementação do componente *TransferMechanism* pode ser adaptada para o padrão ou protocolo específico de cada mecanismo de transferência escolhido.

Dessa forma, a implementação do AIT*m* fica livre para tratar de cada uma das quatro principais ações do processo de *handoff* da forma que achar mais viável ou possível. Ou seja, o leque de aplicações Web suportadas, de eventos que dão inicio ao *handoff*, de métodos para decidir o dispositivo alvo e de mecanismos para a transferência do estado da sessão é tão grande quanto a quantidade de implementações fornecidas para as interfaces.

CAPÍTULO 4

Implementação

Uma implementação da abordagem proposta foi realizada a fim de obter resultados para sua avaliação. O AIT*m* foi implementado como uma extensão para navegador Web na linguagem JavaScript. Além da implementação do AIT*m*, também foi construída uma ferramenta na forma de aplicação Web para configuração das preferências do usuário para a escolha do dispositivo alvo no processo de *handoff*. Essa aplicação foi implementada utilizando as linguagens HTML, CSS e JavaScript. Também foi construída uma aplicação para a plataforma Android que pode ser utilizada para disparar o processo de *handoff*. O código da implementação do protótipo pode ser encontrado em https://github.com/vinicius-lima/ait-middleware.

4.1 Visão geral do ambiente de implementação

A Figura 4.1 ilustra o ambiente de implementação da abordagem proposta. O AIT*m* é implementado como uma extensão que pode ser adicionada em qualquer navegador. Neste trabalho foi escolhido o navegador Mozilla Firefox como ambiente de implementação.

Cada navegador Web executa sobre um dispositivo conectado à Internet. Esses dispositivos podem estar conectados na mesma rede privada ou não. Na Figura 4.1, o *laptop* e o computador de mesa estão na mesma rede privada. Já o *smartphone* está conectado em uma rede diferente. O sistema operacional desses dispositivos pode ser qualquer um que suporte um navegador Web. Por exemplo, um possível sistema operacional para o *smartphone* é uma distribuição do Android. Possíveis sistemas operacionais para o *laptop* e o computador de mesa são uma distribuição Linux ou Windows.

Além da comunicação direta entre os dispositivos na mesma rede, serviços Web externos podem ser utilizados como intermediários durante o processo de *handoff*. Um serviço Web deve oferecer pelo menos uma das seguintes funcionalidades: uma forma de armazenamento de dados, que é utilizada para armazenar o estado da sessão de interação da aplicação; um método para troca de mensagens entre os clientes conectados a ele, que

é utilizado para notificar a troca de dispositivos. No ambiente deste trabalho é utilizado o Firebase [29], que oferece ambas a funcionalidades.

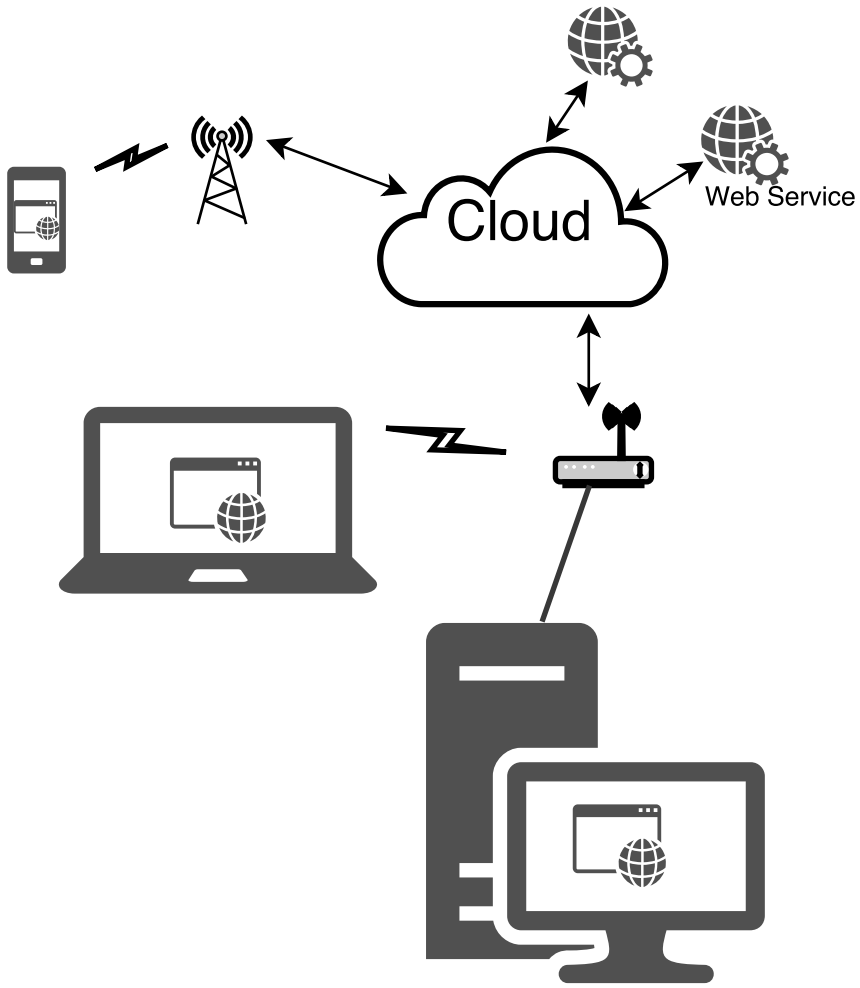

Figura 4.1: *Ambiente de implementação.*

O usuário interage com uma única aplicação Web em um único navegador Web por vez. Durante o processo de *handoff*, o AIT*m* obtém o estado da sessão de interação da aplicação Web executando sobre o navegador Web fonte e envia esse estado através do mecanismo de transferência. No navegador Web alvo, o AIT*m* recebe o estado da sessão e o restaura na aplicação Web.

A captura e restauração do estado da sessão de interação da aplicação Web é feita em nível de sistema, através da troca de mensagens entre o AIT*m* e navegador Web. O evento que dá início ao processo de *handoff* e sua notificação são descritos na seção 4.2. O método utilizado para a escolha do dispositivo alvo do *handoff* é descrito na seção 4.3. Os passos de captura e restauração são detalhados na seção 4.4. As decisões de implementação feitas sobre o mecanismo de transferência são descritas na seção 4.5.

4.2 Disparo do processo de *handoff*

No protótipo implementado, o evento escolhido é a mudança do local físico do usuário. Esse evento é disparado por um botão inserido na interface gráfica do navegador Web pela extensão ou por uma aplicação para a plataforma Android. Ao pressionar o botão inserido na interface gráfica do navegador ou ao abrir-se a aplicação Android, uma lista de locais é apresentada, estes locais são comuns aos de uma casa, por exemplo, quarto, banheiro, sala de estar, escritório, garagem, etc. Um local então é escolhido, esse local é o atual local físico do usuário e de onde um dispositivo alvo deve ser escolhido. Por exemplo, se o usuário estava na sala de estar e chega ao escritório, então o local escolhido na lista será "Escritório" e o botão intitulado "Handoff" é pressionado a fim de dar início ao processo de *handoff*.

A notificação do evento é feita com o suporte da funcionalidade *publish/subscribe* oferecida pelo Firebase. Após o botão "Handoff" ser pressionado, é enviada uma mensagem ao Firebase com a localização física atual do usuário. O Firebase se encarrega de notificar essa mudança às instâncias do AIT*m* que estão em execução nos dispositivos do usuário previamente registrados. Assim, os dispositivos ficam cientes de quando o processo de *handoff* deve ser iniciado e do dado ligado ao evento, ou seja, o local físico do usuário, que é uma das informações utilizada para decidir o dispositivo alvo do *handoff*.

Note que o evento de início do processo de *handoff* não é o apertar de um botão, e sim a mudança do local físico do usuário. Escolher um local em uma lista e apertar um botão para enviar uma mensagem para o Firebase foi o método utilizado para simular essa mudança de localização do usuário. Essa percepção da mudança do local físico do usuário poderia ser feita, por exemplo, com sensores de presença.

4.3 Escolha do alvo do processo de handoff

Uma vez o gatilho disparado, o processo de *handoff* é iniciado, completando a primeira ação do processo de *handoff*. Em seguida, é necessário decidir qual dispositivo do usuário será responsável por dar continuidade a interação do usuário com a aplicação Web. No protótipo implementado, são utilizadas as preferências do usuário para escolher o dispositivo alvo do processo de *handoff*.

O componente *Selector* espera que o *Controller* mande a localização do usuário como entrada, então retorna um identificador do dispositivo escolhido como alvo, como descrito na seção 3.2.4. A partir das preferências, um dispositivo nessa localização é escolhido como alvo do *handoff*. As preferências do usuário e as informações dos dispositivos devem estar previamente registrados. O registro das preferências é feito através de uma aplicação Web oferecida junto com o protótipo do AIT*m*. Esses registros são armazenados

no sistema de armazenamento do Firebase, utilizado como implementação do componente *Storage*.

A Figura 4.2 traz um exemplo de cenário para ilustrar o propósito das preferências do usuário.

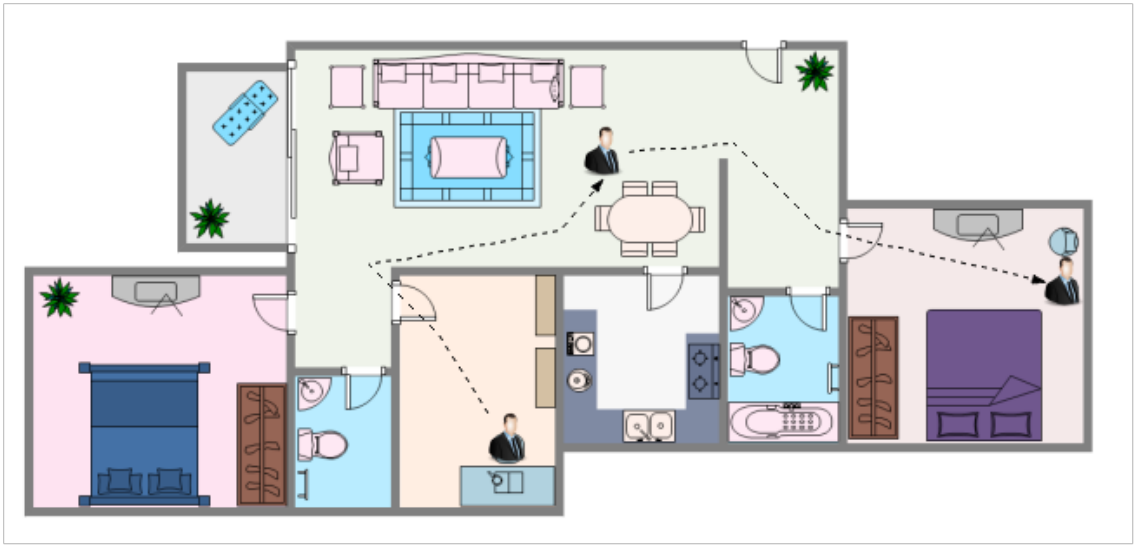

Figura 4.2: *O usuário mantém a interação com a aplicação enquanto se move pela residência.*

As preferências são escritas na forma de regras com respeito aos dispositivos e aplicações utilizados pelo usuário. Um possível conjunto de regras produzidas pelo usuário no cenário da Figura 4.2 poderia ser:

- "Se eu estiver no meu escritório, então eu prefiro utilizar o computador de mesa".
- "Se eu estiver no meu quarto, então eu prefiro utilizar meu *laptop*".
- "Caso contrário, está tudo bem utilizar meu *smartphone*".

Na abordagem proposta, as preferências do usuário são armazenadas em um arquivo JSON (*JavaScript Object Notation*) com a estrutura apresentada abaixo:

Estrutura das preferências no arquivo JSON

```
{
  "application": "applicationID", // Numerical identifier or URL
  "preferences": [
    {
      "location": "current location", // User's physical location
      "device": ["device characteristic", "device characteristic", ...]
    },
    ...
  ]
}
```

Um objeto JSON é estruturado no formato chave–valor (*key–value*), sendo que chave é chamada de propriedade (*property*). Cada aplicação possui seu conjunto de preferências, o qual é estruturado como um objeto JSON. Esse objeto possui duas propriedades, *application* e *preferences*. A primeira propriedade é um identificador único para a aplicação à qual as preferências pertencem, enquanto a segunda propriedade refere-se às preferências em si.

Essas preferências são organizadas no formato de um vetor de objetos JSON, sendo que cada objeto possui duas propriedades, *location* e *device*. A primeira propriedade refere-se ao local físico no qual essas preferências devem ser consultadas. A segunda propriedade é um vetor de características dos dispositivos presentes nesse local físico com os quais o usuário prefere interagir. No vetor da propriedade *device* são armazenadas strings como: id = 5, battery = false, largest screen size, etc. Com essa representação, sempre que necessário, é possível adicionar preferências para novas aplicações e adicionar preferências para diferentes localizações para a mesma aplicação.

Para o protótipo implementado, são suportadas as seguintes *strings* que descrevem as preferências do usuário em relação a um dispositivo: *id*, identificador único do dispositivo do usuário; *type*, tipo de dispositivo que o usuário prefere utilizar, por exemplo, *smartphone*; *battery*, o dispositivo depende ou não de uma bateria; *keyboard*, o dispositivo possui ou não um teclado físico; *width*, largura desejada da tela do dispositivo; *height*, altura desejada da tela do dispositivo.

Abaixo, encontra-se um exemplo de uma possível instância das preferências do usuário em relação a uma aplicação, a dois locais físicos e aos dispositivos presentes nesses locais.

Exemplo de preferências para uma aplicação

```
{
  "application": "github.com",
  "preferences": [
    {
      "location": "Office Room",
      "device": ["width = largest", "battery = false"]
    },
    {
      "location": "Bedroom",
      "device": ["id = 150"]
    }
  ]
}
```

Assim, o componente *Selector* consulta as preferências do usuário para determinar o dispositivo que será responsável por dar continuidade à execução da aplicação

Web. É aplicada uma sequência de filtros para as preferências relacionadas ao local físico passado como parâmetro e a aplicação Web que está sendo utilizada pelo usuário no momento do *handoff*. Cada filtro avalia uma *string* do vetor da propriedade *device*. Os dispositivos que passam por um filtro são avaliados pelo próximo filtro, e assim sucessivamente até todas as características de dispositivos serem avaliadas. No fim, haverá um ou mais dispositivos restantes no vetor e, caso haja mais de um, se escolhe o da primeira posição.

4.4 Obter e restaurar o estado da sessão de interação

Com o dispositivo alvo definido, é o momento de obter o estado de sessão de interação da aplicação Web. No protótipo deste trabalho, o estado da sessão de interação obtido pela implementação do componente *StateManagerImplementation* é resultado da coleta dos dados de três partes da aplicação Web. Essa coleta é feita em nível de sistema, onde o *StateManagerImplementation* troca mensagens com o navegador Web, sem a necessidade de modificação do código fonte da aplicação.

Inicialmente, são coletados os campos do DOM (*Document Object Model*), por exemplo, áreas de texto ou campos de um formulário. O DOM das aplicações Web é estruturado no formato de uma árvore e o componente *StateManagerImplementation* realiza um caminhamento da esquerda para a direta em pré-ordem sobre essa estrutura.

Em seguida, são coletadas as variáveis JavaScript definidas pela aplicação Web. O *StateManagerImplementation* faz uma iteração sobre a variável *window* e coleta os dados das variáveis da aplicação Web.

Finalmente, são coletados os *cookies* do domínio dessa aplicação Web. A coleta dos *cookies* é feita por uma chamada de função da API (*Application Programming Interface*) fornecida pelo navegador. A união dos dados resultantes das três coletas forma o estado da sessão de interação da aplicação Web.

Após a coleta, os dados são serializados em uma *string* no formato JSON e encaminhados para o mecanismo de transferência. A serialização e a desserialização são feitas utilizando o *JSON–cycle*, uma implementação fornecida por Valery Barysok e Douglas Crockford. O *JSON–cycle* trata referências cíclicas que possam ocorrer nas variáveis utilizadas pela aplicação Web.

No dispositivo alvo, a *string* é recuperada do mecanismo de transferência e desserializada, os *cookies* do domínio da aplicação são armazenados no navegador e o código da aplicação Web é carregado através de sua URL. Logo após o código da aplicação ser carregado, são inseridos os dados das variáveis JavaScript e, por fim, os dados do DOM.

O AIT*m* apenas realiza o *handoff* quando as conexões utilizadas pela aplicação não estão transferindo dados, ou seja, na atual implementação o estado de conexões não é migrado. As conexões no dispositivo fonte são encerradas e, após o término do *handoff*, são abertas novas conexões no dispositivo alvo que serão utilizadas pela aplicação. Vale lembrar que as conexões utilizadas pelas aplicações para transferir dados são diferentes das conexões utilizadas pelo AIT*m*.

Caso a aplicação Web utilize algum método de autenticação de usuário e os dados coletados do DOM, das variáveis criadas pela aplicação e dos *cookies* não forem informação suficiente para manter o usuário autenticado, então o usuário não estará autenticado na aplicação no dispositivo alvo. Entretanto, essa barreira pode ser superada oferecendo novas implementações para o componente *StateManagerImplementation* que tratem as especificidades de cada método de autenticação, como mencionado na Seção 3.3.

4.5 Mecanismo de Transferência

Com o estado de sessão de interação da aplicação Web recuperado e com o dispositivo alvo do *handoff* decidido, é o momento de efetivamente enviar os dados do estado da sessão do dispositivo fonte para o alvo. No protótipo deste trabalho, são utilizados três mecanismos diferentes para implementar o componente *TransferMechanism*: *WebSockets*, um mecanismo para comunicação direta entre os dispositivos fonte e alvo do *handoff*; Firebase, um serviço hospedado em um ambiente de nuvem; CYCLON, uma implementação do protocolo de *gossiping*.

WebSocket é uma tecnologia que permite a comunicação bidirecional por canais *full-duplex* sobre um único *socket* TCP (*Transmission Control Protocol*) [10]. Na implementação com *WebSockets*, durante o processo de *handoff*, no dispositivo alvo é criada uma instância do *WebSocket* que irá esperar pelo envio do estado da sessão de interação através da rede. No dispositivo fonte, é criada uma instância do *WebSocket* que se conecta ao dispositivo alvo. Uma vez conectados, o *Controller* utiliza a interface *StateTransfer* para encaminhar ao *TransferMechanism* a *string* criada pelo *StateManagerImplementation*. Assim, o *TransferMechanism* envia o estado da sessão através da conexão. Dessa forma, é possível enviar o estado de sessão entre os dispositivos participantes do *handoff* sem a necessidade de um serviço intermediário.

Na implementação com o Firebase, é utilizada uma API fornecida pelos desenvolvedores do serviço. Os dados no Firebase são armazenados no formato chave–valor. Chave é a URL para acessar os dados, valor são os dados propriamente ditos. Com a API é possível criar referências para as URL's onde podem ser lidos e escritos os dados. Assim, é criada uma referência para a URL na qual o estado da sessão de interação será escrito

e lido durante o processo de *handoff*. Essa referência é criada em todos os dispositivos previamente registrados pelo usuário, entretanto, no momento do *handoff*, apenas o dispositivo fonte escreve nessa referência e apenas o dispositivo alvo realizará a restauração do estado da sessão. Dessa forma, o *Controller* utiliza a API para encaminhar ao Firebase a *string* criada pelo *StateManagerImplementation*. O Firebase então envia o estado da sessão de interação para o dispositivo alvo do *handoff*.

Na implementação com o CYCLON, são utilizada requisições *RESTful* para trocar mensagens entre o *Controller* e o *TransferMechanism*. Como mencionado na Seção 2.4.2, o CYCLON executa em uma camada independente do AIT*m* e cada dispositivo do usuário é um nó na rede mantida pelo CYCLON. Durante o processo de *handoff*, o *Controller* no dispositivo fonte faz uma requisição PUT ao CYCLON a fim de encaminha a *string* criada pelo *StateManagerImplementation*. O CYCLON então se encarrega propagar os dados do estado da sessão entre os nós da rede de acordo com o protocolo implementado por ele. No dispositivo alvo do *handoff*, o *Controller* faz uma requisição GET a fim de recuperar o estado da sessão de interação e então restaurá-lo na aplicação.

4.6 Considerações

Este capítulo apresentou uma possível implementação da abordagem proposta. A mudança do local físico do usuário é utilizada como evento para dar início ao processo de *handoff*. A escolha do dispositivo alvo do *handoff* é feita a partir das preferências do usuário. Para cada aplicação e para cada local, é escolhido o dispositivo que mais atende as preferências do usuário. O estado da sessão de interação é obtido e restaurado em nível de sistema, enviando mensagens entre o AIT*m* e o navegador Web. São utilizados três mecanismos de transferência, cada um deles com seus próprios métodos para a troca de mensagens com o *Controller* e para enviar os dados entre os dispositivos do usuário.

O protótipo implementado suporta aplicações Web que podem ter ser estado de sessão de interação obtido e restaurado a partir dos dados do DOM, das variáveis JavaScript e dos *cookies* de navegador. Caso a aplicação Web necessite de mais informações ou precise seguir algum fluxo específico para restaurar o estado da sessão, então o protótipo pode ser adaptado, oferecendo novas implementações para a interface *StateTransfer*.

Novos eventos de início do *handoff*, novos métodos para a seleção do dispositivo alvo do *handoff* e novos mecanismos de transferências podem ser removidos ou adicionados adaptando-se as implementações das interfaces responsáveis por essas ações. Assim, apesar do protótipo para este trabalho suportar um conjunto de aplicações Web, a implementação pode ser adaptada para suportar qualquer aplicação Web de interesse.

CAPÍTULO 5

Avaliação

Durante o processo de *handoff* os dados do estado da sessão de interação são enviados através da rede entre o dispositivo fonte e o dispositivo alvo. Este capítulo apresenta uma avaliação de como a escolha do mecanismo de transferência do estado da sessão de interação da aplicação Web influencia no tempo para completar o processo de *handoff*. Os mecanismos avaliados são os apresentados na Seção 4.5.

5.1 Ambiente de experimentação

As avaliações realizadas neste trabalho foram feitas em duas máquinas, um computador de mesa e um *laptop*. O navegador Web utilizado foi o Mozilla Firefox e dentre as aplicações Web utilizadas estão o Facebook e o GitHub. As especificações de ambas as máquinas são apresentadas a seguir:

- Especificações do computador de mesa:
  - Processador: 4x AMD Phenom(tm) 9600B Quad-Core Processor.
  - 4 GB de memória RAM.
  - Sistema Operacional: Ubuntu 14.04.05 LTS.
  - Versão do Kernel Linux: 3.13.0-117-generic (x86_64).
  - Compilador Defaut para C: GNU C Compiler versão 4.8.4 (Ubuntu 4.8.4-2ubuntu1∼14.04.3)
  - Placa de Rede: Ethernet - NetXtreme BCM5754 Gigabit Ethernet PCI Express - Broadcom Corporation.
- Especificações do *laptop*:
  - Processador: Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz
  - 8 GB de memória RAM.
  - Sistema Operacional: Windows 10 64bits, processador com base em x64
  - Placa de Rede: Dell Wireless 1707 802.11b/g/n (2.4GHz)

5.2 Envio utilizando um mecanismo de comunicação direta e um serviço na nuvem

A Tabela 5.1 apresenta uma análise do tempo necessário para enviar os dados de estado de uma sessão de interação de uma aplicação entre os dispositivo fonte e alvo do *handoff* utilizando um mecanismo de comunicação direta (WebSocket) e um serviço em nuvem (Firebase). A primeira coluna da tabela representa o tamanho, em megabytes, do estado de sessão enviado entre os dispositivos. Os valores vão de 0,1MB (100kB) até 5MB. Entre as aplicações Web avaliadas, o Facebook teve um estado coletado de aproximadamente 50kB e o GitHub teve um estado coletado de aproximadamente 3kB. Para cada tamanho, foi realizado o envio do estado de sessão por 10 vezes, sendo extraída a média aritmética do tempo necessário para completar o envio, o que é apresentado nas segunda e terceira colunas da tabela.

Tamanho do estado (MB)	Tempo de transferência em rede local (seg)	Tempo de transferência por serviço em nuvem (seg)
0,1	0,0525	5,9528
0,25	0,1368	9,4327
0,5	0,2527	13,2789
1	0,4677	24,4977
2	0,766	43,3573
3	1,2493	70,6231
4	1,603	94,248
5	2,1	120,8527
Desvio padrão	0,750227447	43,08245164

Tabela 5.1: *Tempo de transferência local e pela nuvem*

Com o mecanismo de envio direto, apenas os dispositivos fonte e alvo do *handoff* então envolvidos na transferência. Fazendo uso de um serviço em nuvem, há um terceiro elemento envolvido, o Firebase. Para *WebSockets*, 100kB levam em média 0,05 segundos para serem transferidos e 5MB levam em média 2 segundos para serem transferidos. Para o Firebase, 100kB levam em média 6 segundos e 5MB levam em média 120 segundos para serem transferidos.

O tempo de transferência aumenta de acordo com o aumento do tamanho do estado da sessão enviado e, como esperado, a média de tempo para uma transferência local é menor do que a média de tempo da transferência que faz uso de um serviço em nuvem. Um dos motivos desse comportamento é devido ao serviço na nuvem fazer uso da rede pública, a Internet, para enviar os dados. Nessa rede os pacotes realizam mais saltos

para chegarem aos seus destinos, seja ele o serviço na nuvem ou o dispositivo alvo do *handoff*. Como a rede é pública, também é compartilhada com outros usuários e serviços, o que aumenta o atraso para os pacotes percorrem suas rotas.

Outro motivo é a forma como cada mecanismo de transferência é implementado e o protocolo de comunicação utilizado por cada um deles. Na implementação de *WebSockets* utilizada para os testes, são apenas abertos *sockets* TCP em cada um dos dispositivos participantes do *handoff*, então, o estado da sessão é enviado por essa conexão. O protocolo em nível de aplicação é simples, as verificações de erro e garantia de entrega utilizadas são as já oferecidas pelo TCP. Há apenas dois participantes na conexão e ela é criada com o único propósito de enviar os dados do estado da sessão.

A implementação do Firebase, assim como o protocolo utilizado por ele, são mais complexos. O Firebase é um serviço de armazenamento de dados e troca de mensagens genérico, ou seja, visa abranger a maior quantidade possível de usuários. Cada usuário utiliza o serviço para um propósito diferente, exigindo que o Firebase seja robusto o suficiente para atender a cada usuário. Para ser robusto, é necessário implementar em nível de aplicação verificações, rotinas e funções que permitem o Firebase suportar seus usuários.

Por exemplo, para garantir que um dado seja efetivamente escrito, o Firebase utiliza a abordagem de *best–effort* para que, sempre que a requisição de escrita não for bem sucedida, então, após um intervalo de tempo, a requisição de escrita é feita novamente. Outro exemplo é, durante a escrita no Firebase, caso haja algum dado escrito anteriormente, é avaliado o que mudou do dado anterior para o que está sendo escrito nesse momento. Assim, apenas a diferença é escrita. Realizar essa ação também aumenta no tempo total para que as referências a esse dado sejam notificadas da mudança. Essas referências são as citadas na seção 4.5.

5.3 Envio utilizando um protocolo de comunicação baseado em gossiping

A fim de comparar outras possibilidades para mecanismos de transferência além do simples uso de um modelo cliente–servidor, este trabalho utiliza um protocolo de comunicação baseado no conceito de *gossiping* chamado CYCLON [31]. Esse protocolo realiza *multcast* para enviar os dados, assim, um nó não envia dados para todos os outros nós da rede e sim para um conjunto de nós vizinhos.

No protocolo CYCLON, cada nó irá periodicamente realizar o envio de seus dados a fim de que a rede convirja, ou seja, para que todos os nós da rede tenham os mesmos dados. No caso deste trabalho, esses dados referem-se ao estado da sessão

de interação da aplicação. Na implementação do protocolo CYCLON utilizada por este trabalho, essa periodicidade do envio dos dados pode ser estabelecida pelo parâmetro RPS *period*.

A Figura 5.1 apresenta o tempo necessário para que todos os nós da rede possuam o mesmo estado da sessão de interação da aplicação. Foram avaliadas uma rede com 5 nós e uma com 20 nós. Os valores escolhidos para o parâmetro RPS *period* foram, 150, 300 e 600 milissegundos. Cada valor foi testado 10 vezes durante 300 segundos e o tamanho do estado da sessão foi fixo durante toda a avaliação.

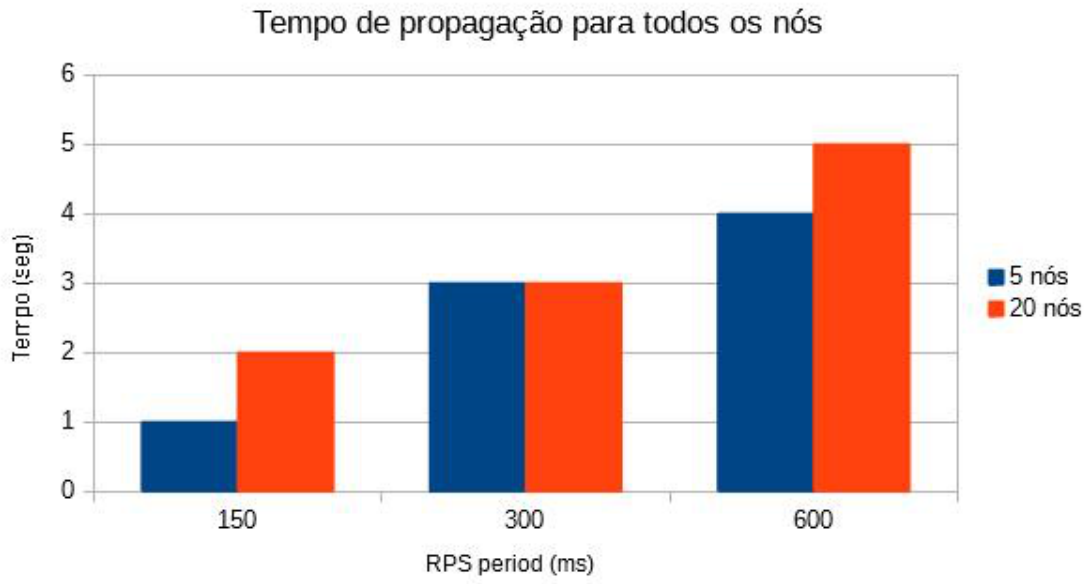

Figura 5.1: *Tempo para o estado estar em todos os nós*

Quanto menor o valor para o RPS *period*, menor é o tempo para que a rede convirja. De mesma forma, quanto maior o valor para RPS *period*, maior é o tempo para que a rede convirja. Como apresentado em 2.4.2, em um protocolo baseado em *gossiping*, um nó envia dados apenas para seus vizinhos. Esses vizinhos são escolhidos a partir de uma lista de pares. A construção e atualização dessa lista, bem como a quantidade de nós contidos nela, depende da implementação do protocolo de *gossiping*.

Assim, a escolha do valor para o parâmetro RPS *period* dependerá de quanto tempo a aplicação pode esperar para que o processo de *handoff* seja concluído. Por exemplo, se o usuário está indo de uma sala para outra, 5 segundos pode ser um tempo aceitável para o *handoff*. Agora, se o usuário apenas mudou de acento na mesma sala ou apenas mudou de dispositivo sem se mover do lugar, então a aplicação irá exigir um tempo menor para que o *handoff* seja concluído.

5.4 Considerações

A escolha entre um serviço intermediário na nuvem e um mecanismo de comunicação direta tem reflexo no tempo total para realizar o processo de *handoff*, pois o serviço na nuvem não estará na mesma rede privada que a dos dispositivos envolvidos no *handoff*. O tempo para concluir o *handoff* também muda dependendo de como o serviço é implementado e do protocolo de comunicação utilizado por ele. Entretanto, o uso de um serviço na nuvem garante resiliência, uma vez que, caso algum erro ocorra com o dispositivo fonte durante o processo de *handoff*, o estado de sessão da aplicação Web poderá ser recuperado a partir da cópia armazenada no serviço na nuvem.

Com o protocolo de comunicação baseado em *gossiping*, pode-se utilizar as vantagens do mecanismo de transferência local e a vantagem do armazenamento redundante do estado da sessão, pois cada nó irá manter uma cópia. O protocolo CYCLON também se preocupa em manter a conectividade da rede e realiza reparos cada vez que um nó entra ou sai da rede, aumentando a resiliência da abordagem. O número de mensagens de controle enviadas por um nó não é influenciado pela entra ou saída de nós da rede, o que ajuda na escalabilidade da implementação.

Esses pontos de resiliência também são a desvantagem de protocolos baseados em *gossiping*, pois mesmo os dispositivos que não são a fonte ou alvo do processo de *handoff* irão manter uma cópia do estado da sessão. Como periodicamente esses dados são enviados, os nós da rede irão periodicamente consumir recursos, o que pode tornar o dispositivo inapto para uso caso possua recursos limitados, como, carga de bateria.

CAPÍTULO 6

Trabalhos Relacionados

Este capítulo apresenta os trabalhos que abordam o problema de migração de aplicações utilizando diferentes técnicas.

6.1 Abordagem baseada em percepção de contexto

Siu *et al.* [26] afirmam que utilizar um estado de execução capturado sem nenhum processamento de adaptação para o dispositivo alvo não é viável para ambientes ubíquos por serem heterogêneos. Para esses ambientes, deve haver uma gerência na captura e restauração do estado de execução da aplicação, ou seja, a gerência deve ser ciente do contexto (*context-aware*). Assim, um mecanismo de migração de estado foi implementado no *Sparkle Pervasive Computing Enviroment*, uma plataforma para a execução de aplicações ubíquas.

Para toda aplicação executando no *Sparkle*, seu estado é centralizado em um contêiner. Dessa forma, durante a migração, apenas o estado nesse contêiner precisa ser capturado, processado, migrado e restaurado. No sistema *Sparkle*, aplicações são compostas de *facets*. *Facets* são unidades funcionais independentes dos dados ou interface com usuário. Cada *facet* possui um único método público, não interage com o usuário e não mantém dados sobre o estado da aplicação. Cada *facet* é associada a um arquivo que contém a descrição dos seus recursos necessários, de seu comportamento em tempo de execução, de suas dependência de contexto, etc. Cada aplicação é associada a um contêiner. Esse contêiner contém a interface com o usuário e armazena os dados e estado de execução da aplicação. O contêiner também armazena o conjunto de funcionalidades oferecidas pela aplicação.

O sistema *Sparkle* possui módulos que interagem entre si a fim de prover suporte à migração ciente de contexto de aplicações. Os módulos são: o Gerenciador de *Facets*, o Gerenciador de Dados do Estado da Aplicação, o Gerenciador de Aplicações, o Gerenciador do Estado do Contexto, o Gerenciador Central e o Gerenciador de Mobilidade. O Gerenciador de *Facets* recupera estatísticas de uso dos *facets*. Essa informação ajuda o Gerenciador de Mobilidade realizar o pré-carregamento de *facets* (no lado do destino)

quando a aplicação está migrando. Quando há uma requisição de migração, o Gerenciador de Dados do Estado da Aplicação fica responsável por capturar o estado da aplicação, ou seja, o estado do contêiner. O Gerenciado de Aplicações atua como intermediário entre o contêiner e a interface com o usuário. Essa intermediação facilita ao programador gerenciar a comunicação entre os componentes gráficos da interface.

O contexto do ambiente ou as condições da máquina irão mudar com o tempo. O Gerenciador do Estado do Contexto recupera informações sobre o contexto da aplicação periodicamente. Exemplo de informações coletadas são: tempo atual, localização, condições da máquina, etc. O Gerenciador Central, como o nome sugere, é a entidade central do sistema *Sparkle*. Ele é responsável por coordenar as atividade dos outros módulos do sistema. O Gerenciador de Mobilidade é a principal parte do sistema de mobilidade. Ele verifica se o usuário necessita migrar e, caso necessite, ele irá decidir o destino. O Gerenciador de Mobilidade também realiza o processamento do estado capturado antes e após a migração da aplicação.

No mecanismo implementado por Siu et al. [26], o sistema *Sparkle* funciona em cinco fases lógicas: Captura do Estado (na fonte), Processamento do Estado (na fonte), Transmissão do Estado, Processamento do Estado (no destino) e Restauração do Estado (no destino). Na captura do estado, o Gerenciador de Dados do Estado da Aplicação captura o estado da aplicação e gera um arquivo XML. Os dados do estado captura são divididos em três categorias: tipos fixos, dados desse tipo não mudam mesmo que o contexto mude; tipos mutáveis, dados desse tipo se ajustam dependendo do contexto; tipos descartáveis, dados desse tipo são removidos a fim de diminuir o tamanho do arquivo XML. Um exemplo para o tipo mutável é resolução de tela, que pode reduzir de 1400x1050 para 320x480.

No primeiro processamento do estado feito pelo Gerenciador de Mobilidade na fonte, os dados desnecessários que foram capturados são descartados. Na transmissão do estado, o estado é enviado através da rede para o alvo. No segundo processamento do estado feito pelo Gerenciador de Mobilidade no destino, os dados são adaptados ao contexto do alvo. Finalmente, na restauração do estado, o container que mantém os dados do estado de execução da aplicação é reconstruído.

6.2 Abordagem baseada em agentes móveis

Zhou *et al.* [34] utilizam a autonomia e mobilidade de agentes móveis para propor uma arquitetura baseada em agentes chamada *MDAgent*. A abordagem proposta requer que as aplicações sejam construídas ou adaptadas para suportar a arquitetura *MDAgent*. A arquitetura das aplicações suportadas possui duas camadas. A camada superior consiste de componentes da aplicação, como, lógica da funcionalidades da

aplicação, interfaces com o usuário, recursos, etc. Nessa camada superior também são encontrados arquivos descritivos, como, perfis de usuário, perfis de dispositivos, perfis de recursos e descrição de interfaces.

Na camada base, os principais módulos são: o coordenador, o gerenciador de captura de estado, o agente móvel e o adaptador. O coordenador estabelece um link de sincronização entre as interfaces com o usuário e troca mensagens com o gerenciador de captura de estado e o agente móvel. Basicamente, cada interface com o usuário se registra junto ao coordenador. Quando o estado muda, essas interface são notificadas automaticamente. Dessa forma, simplifica o controle de consistência entre as interface e aumenta a reusabilidade de componentes.

O gerenciador de captura de estado é responsável pelo controle do processo de persistência do estado de aplicações em execução, enquanto o agente móvel é responsável por encapsular e migrar o estado da aplicação para o destino. Devido a heterogeneidade dos ambiente de execução, o adaptador lida com a incompatibilidade dos ambientes.

A abordagem é apresentada como uma arquitetura. A arquitetura é composta por quatro camadas: Camada de Sensores, Camada de Contexto, Camada de Agentes e Camada de Aplicação. A Camada de Sensores coleta dados dos sensores físicos ou lógicos que detectam a movimentação do usuário, conexão de rede, latência, etc. A Camada de Contexto armazena os dados coletados pelos sensores e, se alguma condição predefinida aconteça, os agentes autônomo que constituem a aplicação são disparados e iniciam a migração.

A Camada de Agentes é responsável por conectar a Camada de Contexto e a Camada de Aplicação. Essa conexão é feita por dois gerenciadores de agentes, o Gerenciador de Agentes Móveis e o Gerenciador de Agentes Autônomos. O Agente Autônomo é responsável por processar e tomar decisões de acordo com os dados recebidos da Camada de Contexto. O Agente Móvel é responsável por encapsular os componentes da aplicação. Eles se comunicam através da troca de mensagens. Quando o agente autônomo detecta uma possível movimentação do usuário ou o usuário explicitamente indica que irá mudar de dispositivo, o agente autônomo primeiro notifica o agente móvel para se preparar para migrar, então, o agente autônomo recupera o estado da aplicação. Após decidido o destino, o agente móvel irá assumir a transmissão do estado da aplicação.

6.3 Abordagem com instrumentação do código fonte da aplicação Web

Lo *et al.* [17] propõem o *Imagen*. Uma nova técnica e ferramenta para migrar, do lado do cliente, o estado de sessão de aplicações Web entre diferentes navegadores e

dispositivos.

Imagen funciona através da instrumentação do código da aplicação. Essa instrumentação pode ser feita pelo desenvolvedor da aplicação ou pelo usuário final da aplicação. O desenvolvedor pode instrumentar o código da aplicação durante sua construção. Já o usuário final pode habilitar essa instrumentação com o uso de um proxy HTTP que irá executar no dispositivo do usuário. O proxy HTTP é fornecido pelos criadores do *Imagen*.

Utilizando um botão na interface gráfica, inserido pela instrumentação do código da aplicação Web, o usuário pode capturar o estado da aplicação a qualquer momento durante a execução. Os dados do estado capturado são salvos em um armazenamento secundário, podendo ser um serviço Web ou o disco local do usuário. O usuário então recebe uma URL que será utilizada para recuperar e carregar o estado da aplicação Web no navegador no dispositivo alvo.

No navegador do dispositivo alvo, o usuário insere a URL que lhe foi previamente dada. Se o código foi instrumentado pelo desenvolvedor, então não é necessário o uso do proxy HTTP. Caso contrário, o proxy redireciona o navegador do usuário para a URL original de onde o estado foi capturado. Entretanto, ao invés de retornar o conteúdo da URL original, é retornado o estado que foi salvo. Essa abordagem permite que a aplicação restaurada execute no mesmo domínio de segurança da aplicação original. Após a aplicação estar carregada no novo navegador, ela continua a executar no ponto no qual parou.

6.4 Abordagem sem instrumentação do código fonte da aplicação Web

Oh *et al.* [21] propõem um arcabouço para que aplicações Web possam migrar sua execução entre navegadores. O estado é salvo na forma de um *snapshot*, que na verdade é outra aplicação Web composta por HTML, CSS e JavaScript, e, se executado o *snapshot*, o estado da aplicação será restaurado automaticamente.

A atual implementação é oferecida na forma de uma extensão para navegadores Web. Dessa forma, a abordagem de Oh *et al.* não utiliza servidores proxy para instrumentar o código da aplicação Web. O estado da aplicação é salvo ao apertar o botão inserido pela extensão de navegador.

Durante a execução da aplicação, se o usuário deseja salvar o atual estado da aplicação Web, ele clica no botão para executar a extensão é salvar o estado. O estado capturado é salvo em uma arquivo *snapshot*, que será transferido para um dispositivo diferente.

A restauração da aplicação realiza um processo similar. O usuário carrega o código da aplicação Web pressionando seu ícone ou inserindo sua URL. Então, o usuário clica no botão inserido pela extensão para restaurar o estado de execução da aplicação. O arquivo *snapshot* é lido do sistema de armazenamento local e executado, o que restaura o estado de execução da aplicação.

6.5 Considerações

A abordagem de Siu *et al.* [26] lida com a captura e restauração do estado da aplicação e com a transferência dos dados do estado. O estado da aplicação é restaurado e adaptado para o contexto do dispositivo que dará continuidade a execução da aplicação. Entretanto, não há espaço para inserir novos métodos para a escolha do dispositivo alvo da migração. O início da migração e a escolha do dispositivo alvo são feito explicitamente pelo usuário durante migração.

A abordagem de Zhou *et al.* [34] abrange os eventos que podem dar início a migração, abrange a captura e restauração do estado da aplicação e abrange os mecanismos de transferência dos dados do estado da aplicação. Entretanto, não abrange os possíveis métodos para escolha do dispositivo alvo da migração, sendo explicitamente escolhido pelo usuário. As aplicações suportadas pela abordagem de Zhou et al. devem ser construídas ou adaptadas para a arquitetura de aplicação definida por eles, pois o estado é capturado e restaurado em nível de aplicação.

A abordagem de Lo *et al.* [17] trata apenas da captura e restauração o estado de aplicações Web, não apresentado métodos para tratar as outras três principais ações do *handoff* e, consequentemente, como elas devem interagir entre si. A abordagem de Lo *et al.* exige instrumentação do código fonte da aplicação e abrange um leque maior de exceções e especificidades do processo de captura e restauração do estado de aplicações Web do que a abordagem proposta no trabalho descrito nesta dissertação. Assim, o trabalho de Lo *et al.* pode ser visto como mais uma possível implementação para a interface *StateManager*.

A abordagem de Oh *et al.* [21] também trata apenas da captura e restauração do estado da aplicação Web, entretanto, não é necessário a instrumentação do código fonte da aplicação. O estado da aplicação é obtido e restaurado em nível de sistema por uma extensão de navegador.

A nossa abordagem, o AIT*m*, abrange as quatro principais ações necessárias para realizar por completo o processo de *handoff* do estado de sessão de interação de aplicações Web. Ela não limita o conjunto de eventos que podem dar início ao processo de *handoff*, sendo possível adaptar o AIT*m* através da implementação da interface *HandoffTrigger*.

Uma indicação explicita do dispositivo alvo do *handoff* feita pelo usuário é uma possível implementação para decidir o dispositivo alvo, mas não a única. O AIT*m* permite o uso de diferentes métodos para decidir o dispositivo alvo do *handoff*, sem exigir que o usuário execute alguma ação que indique explicitamente o dispositivo alvo. Essa escolha pode ser feita de forma automática e pode ser adaptada de acordo com a implementação da interface *TargetSelector*.

O AIT*m* permite que a captura e restauração do estado da aplicação Web sejam feitas em nível de aplicação ou em nível de sistema. Essa decisão é tomada de acordo com a implementação fornecida para a interface *StateManager* e não exige interferência do usuário. Assim, as aplicações Web não são forcadas a ter seu código fonte modificado ou a seguir uma arquitetura de construção de aplicações pré-definida.

O AIT*m* também se adapta de acordo com o mecanismo de transferência escolhido para implementação da interface *StateTransfer*, permitindo que os dados do estado da sessão de interação de aplicações Web sejam enviados entre os dispositivos do usuário.

A Tabela 6.1 resume a comparação entre o AIT*m* e as demais abordagens dos trabalhos relacionados. Na primeira coluna estão os títulos dos trabalhos, sendo o último título a abordagem proposta neste trabalho. As demais colunas representam as quatro funcionalidades necessárias para realizar por completo o processo de *handoff*. Os trabalhos que tratam uma determinada funcionalidade possuem um X para representar essa informação.

Trabalho	Evento de início do handoff	Seleção do dispositivo alvo	Captura e restauração do estado	Mecanismo de transferência
Context-Awere State Managemet for Ubiquitous Applications [26]			X	X
A middleware support for agent-based application mobility in pervasive environments [34]	X		X	X
Imagen: Runtime migration of browser sessions for JavaScript Web Applications[17]			X	
Migration of Web Applications with seamless execution [21]			X	
AIT*m* – Middleware para Migração de Sessões de Interação em Aplicações Web	X	X	X	X

Tabela 6.1: *Comparação com trabalhos relacionados.*

CAPÍTULO 7

Conclusão

Este trabalho apresenta um *middleware* por meio do qual aplicações Web podem adquirir a capacidade de migrar as sessões de interação com o usuário. O estado da sessão de interação de aplicações Web pode ser migrado de um dispositivo para outro de forma automática e transparente, sendo tratadas quatro funcionalidades principais: os eventos que dão início ao processo de *handoff*, a escolha do dispositivo alvo do *handoff*, a captura, armazenamento e restauração do estado de sessão de interação e os mecanismos de transferência.

A abordagem proposta pode ser adaptada para os possíveis métodos que tratam de cada funcionalidade principal do *handoff*. Essa adaptabilidade é alcançada fornecendo diferentes implementações para as interfaces da arquitetura. Um método escolhido para implementar um componente da arquitetura não afeta como outros componentes são implementados e não afeta como o *Contoller* exerce sua função durante o processo de *handoff*.

A escolha do mecanismo de transferência influencia no tempo total necessário para completar o processo de *handoff*. Cada mecanismo de transferência apresenta vantagens e desvantagens como, menor quantidade de saltos na rede entre os dispositivos fonte e alvo do *handoff*, oferta de métodos de resiliência a falhas, menor consumo de recursos devido à menor quantidade de mensagens do protocolo de comunicação, e auto-reparo da rede com a entrada e saída de nós.

7.1 Trabalhos futuros

Surgem oportunidades de trabalhos futuros para a arquitetura proposta neste trabalho. Podem ser feitas extensões para a arquitetura a fim de suportar não apenas aplicações Web, mas todo o conjunto de aplicações ubíquas, independentemente da plataforma de execução.

Atualmente, o *Controller* interage com apenas uma implementação de cada componente que implementa uma interface da arquitetura. Caso seja necessário mudar a implementação de um componente a fim de suportar diferentes especificidades, a

implementação antiga deve ser substituída pela nova e a extensão de navegador dever ser reconstruída e reinstalada.

Por exemplo, se o mecanismo de transferência que está sendo utilizado for de comunicação direta entre os dispositivos e deseja-se mudar para um serviço na nuvem, então os trechos de código do *middleware* que utilizam a comunicação direta devem ser substituídos pelos trechos de código que utilizam o serviço na nuvem. Em resumo, o *Controller* não escolhe em tempo de execução qual implementação deve utilizar para cada interface. Essa escolha é feita no momento em que a extensão é construída.

Uma possível extensão da arquitetura é permitir que o *Controller* escolha em tempo de execução qual implementação dos componentes utilizar para dar suporte as quatro funcionalidades principais do *handoff*. Essas implementações podem ser oferecidas junto com a extensão de navegador ou podem ser armazenadas em um repositório. O repositório será utilizado pelo *Controller* para obter a implementação dos componentes e, cada componente armazenado nele oferecerá uma implementação que dá suporte as diferentes possibilidades de aplicações Web, de eventos, de métodos de seleção do dispositivo alvo e de mecanismos de transferência.

O *Controller* é o componente responsável por garantir que sequência de passos do processo de *handoff* seja realizada, fazendo com que todos os outros componentes do AIT*m* se comuniquem com ele. Dessa forma, o *Controller* se torna um ponto único de falha pois, caso ocorra algum erro nas funções desempenhadas pelo *Controller*, a sequência do processo de *handoff* será interrompida. Um trabalho futuro é avaliar como retomar a sequência de passos do *handoff* caso haja algum erro no *Contoller*.

O escopo do trabalho se manteve em uma aplicação mantendo uma sessão de interação com um único usuário. Caso o usuário mude de dispositivo, então a sessão é encerrada no dispositivo anterior e uma nova é iniciada no dispositivo com o qual o usuário irá interagir. Dessa forma, aplicações colaborativas e multi sessões de usuário estiveram fora do escopo do trabalho. Outra limitação é, caso o usuário tenha mais de uma aba aberta no navegador e cada aba estiver com uma aplicação em execução, apenas o estado de sessão da aplicação que está na aba sendo utilizada pelo usuário no momento do início do *handoff* será transferida. Assim, um trabalho futuro pode abranger também aplicações colaborativas e multi sessões de usuário, bem como a migração do estado de sessão de interação de mais de uma aplicação ao mesmo tempo.

Referências Bibliográficas

[1] AUGUSTIN, I.; YAMIN, A.; GEYER, C. F. **Managing the follow-me semantics to build large-scale pervasive applications**. In: *Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, p. 1–8. ACM, 2005.

[2] BANAVAR, G.; BERNSTEIN, A. **Software infrastructure and design challenges for ubiquitous computing applications**. *Communications of the ACM*, 45(12):92–96, 2002.

[3] BENLIAN, A.; HESS, T.; BUXMANN, P. **Software-as-a-service**. *Wiesbaden: Gabler*, 2010.

[4] BERNSTEIN, P. A. **Middleware: a model for distributed system services**. *Communications of the ACM*, 39(2):86–98, 1996.

[5] BHARDWAJ, S.; JAIN, L.; JAIN, S. **Cloud computing: A study of infrastructure as a service (iaas)**. *International Journal of engineering and information Technology*, 2(1):60–63, 2010.

[6] BRUNEO, D.; PULIAFITO, A.; SCARPA, M. **Mobile middleware: Definition and motivations**. *The Handbook of Mobile Middleware*, p. 145–167, 2007.

[7] CHU, H.-H.; SONG, H.; WONG, C.; KURAKAKE, S.; KATAGIRI, M. **Roam, a seamless application framework**. *Journal of Systems and Software*, 69(3):209–226, 2004.

[8] DE ARAUJO, R. B. **Computação ubíqua: Princípios, tecnologias e desafios**. In: *XXI Simpósio Brasileiro de Redes de Computadores*, volume 8, p. 11–13, 2003.

[9] DRAGO, I.; MELLIA, M.; M MUNAFO, M.; SPEROTTO, A.; SADRE, R.; PRAS, A. **Inside dropbox: understanding personal cloud storage services**. In: *Proceedings of the 2012 ACM conference on Internet measurement conference*, p. 481–494. ACM, 2012.

[10] FETTE, I. **The websocket protocol**. 2011.

[11] FOSTER, I.; ZHAO, Y.; RAICU, I.; LU, S. **Cloud computing and grid computing 360-degree compared**. In: *2008 Grid Computing Environments Workshop*, p. 1–10. Ieee, 2008.

[12] HARTMAN, J.; MANBER, U.; PETERSON, L.; PROEBSTING, T. **Liquid software: A new paradigm for networked systems**. Technical report, Technical Report 96, 1996.

[13] JELASITY, M.; GUERRAOUI, R.; KERMARREC, A.-M.; VAN STEEN, M. **The peer sampling service: Experimental evaluation of unstructured gossip-based implementations**. In: *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, p. 79–98. Springer-Verlag New York, Inc., 2004.

[14] JOHNSTON, A. B.; BURNETT, D. C. **WebRTC: APIs and RTCWEB protocols of the HTML5 real-time web**. Digital Codex LLC, 2012.

[15] JONES, M.; BRADLEY, J.; SAKIMURA, N. **Java web token (jwt)**. RFC 7519, 2015.

[16] KRUMM, J. **Ubiquitous computing fundamentals**. CRC Press, 2016.

[17] LO, J. T. K.; WOHLSTADTER, E.; MESBAH, A. **Imagen: runtime migration of browser sessions for javascript web applications**. *World Wide Web 2013*, p. 815–826, 2013.

[18] MELL, P.; GRANCE, T. **The nist definition of cloud computing**. 2011.

[19] MIKKONEN, T.; SYSTÄ, K.; PAUTASSO, C. **Towards liquid web applications**. In: *International Conference on Web Engineering*, p. 134–143. Springer, 2015.

[20] MIORANDI, D.; SICARI, S.; DE PELLEGRINI, F.; CHLAMTAC, I. **Internet of things: Vision, applications and research challenges**. *Ad Hoc Networks*, 10(7):1497–1516, 2012.

[21] OH, J.; KWON, J.-W.; PARK, H.; MOON, S.-M. **Migration of web applications with seamless execution**. *ACM SIGPLAN Notices*, 50(7):173–185, 2015.

[22] PERERA, C.; ZASLAVSKY, A.; CHRISTEN, P.; GEORGAKOPOULOS, D. **Context aware computing for the internet of things: A survey**. *IEEE Communications Surveys & Tutorials*, 16(1):414–454, 2014.

[23] PINUS, H. **Middleware: Past and present a comparison**. In: *Available: http://userpages. umbc. edu/˜ dgorin1/451/middleware/middleware. pdf*, 2004.

[24] RORIZ, M. P.; MASSARANI, M. A. L.; FREITAS, L. A.; DA ROCHA, R. C. A.; COSTA, F. M. **C3s: A content sharing middleware for smart spaces**. In: *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on*, p. 163–168. IEEE, 2013.

[25] SANAEI, Z.; ABOLFAZLI, S.; GANI, A.; BUYYA, R. **Heterogeneity in mobile cloud computing: taxonomy and open challenges**. *IEEE Communications Surveys & Tutorials*, 16(1):369–392, 2014.

[26] SIU, P. P. L.; BELARAMANI, N. M.; WANG, C.-L.; LAU, F. C. M. **Context-Aware State Management for Ubiquitous Applications.** *Euc*, p. 776–785, 2004.

[27] TAIVALSAARI, A.; MIKKONEN, T.; SYSTÄ, K. **Liquid software manifesto: the era of multiple device ownership and its implications for software architecture**. In: *Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual*, p. 338–343. IEEE, 2014.

[28] TAKASHIO, K.; SOEDA, G.; TOKUDA, H. **A mobile agent framework for follow-me applications in ubiquitous computing environment**. In: *Distributed Computing Systems Workshop, 2001 International Conference on*, p. 202–207. IEEE, 2001.

[29] TAMPLIN, J.; LEE, A. **Firebase**. https://firebase.google.com/, 2016. Último acesso: 04 de Agosto de 2016.

[30] VAN DER MERWE, J.; RAMAKRISHNAN, K.; FAIRCHILD, M.; FLAVEL, A.; HOULE, J.; LAGAR-CAVILLA, H. A.; MULLIGAN, J. **Towards a ubiquitous cloud computing infrastructure**. In: *Local and Metropolitan Area Networks (LANMAN), 2010 17th IEEE Workshop on*, p. 1–6. IEEE, 2010.

[31] VOULGARIS, S.; GAVIDIA, D.; VAN STEEN, M. **Cyclon: Inexpensive membership management for unstructured p2p overlays**. *Journal of Network and systems Management*, 13(2):197–217, 2005.

[32] WEISER, M. **The computer for the 21st century**. *Scientific american*, 265(3):94–104, 1991.

[33] ZHANG, Q.; CHENG, L.; BOUTABA, R. **Cloud computing: state-of-the-art and research challenges**. *Journal of internet services and applications*, 1(1):7–18, 2010.

[34] ZHOU, Y.; CAO, J.; RAYCHOUDHURY, V.; SIEBERT, J.; LU, J. **A middleware support for agent-based application mobility in pervasive environments**. *Proceedings - International Conference on Distributed Computing Systems*, 2007.