UNIVERSIDADE DO VALE DO RIO DOS SINOS - UNISINOS
UNIDADE ACADÊMICA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA INTERDISCIPLINAR DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA
NÍVEL MESTRADO

Marcus Vinicius Lewis Martins

FrameTrail: Um Framework para o Desenvolvimento de Aplicações Orientadas a Trilhas

São Leopoldo
2011

MARCUS VINÍCIUS LEWIS MARTINS

FrameTrail: Um Framework para o Desenvolvimento de Aplicações Orientadas a Trilhas

Dissertação apresentada como requisito parcial para a obtenção do título de Mestre, pelo Programa Interdisciplinar de Pós-Graduação em Computação Aplicada da Universidade do Vale do Rio dos Sinos

Prof. Dr. Jorge Luis Victória Barbosa
Orientador

São Leopoldo, Maio de 2011

M386f Martins, Marcus Vinícius Lewis
FrameTrail: um framework para o desenvolvimento de aplicações orientadas a trilhas / por Marcus Vinícius Lewis Martins. – São Leopoldo, 2011.

90 f. : il. color. ; 30 cm.

Dissertação (mestrado) – Universidade do Vale do Rio dos Sinos, Programa Interdisciplinar de Pós-Graduação em Computação Aplicada, São Leopoldo, RS, 2011.
Orientação: Prof. Dr. Jorge Luis Victória Barbosa, Ciências Exatas e Tecnológicas.

1.Arquitetura de computador – Trilhas. 2.Framework (Programa de computador) – FrameTrail. 3.Agentes inteligentes (software). 4.Sensibilidade ao contexto. I.Barbosa, Jorge Luis Victória. II.Título.

CDU 004.22
004.89

Catalogação na publicação:
Bibliotecária Carla Maria Goulart de Moraes – CRB 10/1252

Marcus Vinicius Lewis Martins

FrameTrail: Um Framework para o Desenvolvimento de Aplicações Orientadas a Trilhas

Dissertação apresentada como requisito parcial para a obtenção do título de Mestre, pelo Programa Interdisciplinar de Pós-Graduação em Computação Aplicada da Universidade do Vale do Rio dos Sinos – UNISINOS

Aprovado em _____/_____/_____

BANCA EXAMINADORA

_____

Prof. Dr. Jorge Luis Victória Barbosa - UNISINOS

_____

Prof. Dr. Luiz Antônio Moro Palazzo - UCPEL

_____

Prof. Dr. Sérgio Crespo Coelho da Silva Pinto - UNISINOS

RESUMO

Este trabalho apresenta o FrameTrail, um *framework* para o desenvolvimento de aplicações orientadas a trilhas. O FrameTrail se propõe a padronizar a manipulação de trilhas independente do seu domínio de aplicação, ou seja, o *framework* disponibiliza uma série de funcionalidades que gerenciam a trilha de uma entidade, sem limitar o tipo de contexto e quais os dados específicos das entidades que as aplicações desenvolvidas com base nele manipulam.

Além de apresentar os componentes internos da arquitetura do FrameTrail, é também apresentada a estrutura de provedores externos que o *framework* disponibiliza para que as aplicações possam tratar de trilhas sem a necessidade de seguir um modelo de tecnologia pré-estabelecido. Através desses provedores cada aplicação pode definir quais são as informações relevantes para os seus contextos, assim como permite a parametrização da maneira de como os seus dados serão armazenados.

A dissertação também descreve o protótipo desenvolvido do FrameTrail, assim como os estudos de caso desenvolvidos para a sua validação e como eles foram direcionados para aproveitar o melhor as funcionalidades providas pelo *framework*.

Palavras-chave: Trilhas, Sensibilidade ao Contexto, *Framework*.

ABSTRACT

This paper presents the FrameTrail, a framework for developing trail oriented applications. The FrameTrail aims to standardize the manipulation of trails regardless of its scope. The framework provides a number of features that manage the entity's trail, without limiting the type of context and what the specifics information of the entities that applications developed based on this can handle.

We present the internal components of the architecture of FrameTrail and also presents the structure of external providers. This external providers provides the framework so that applications can address the trails without the need to follow a technology model pre-established. Each of these providers can define which information relevant to their contexts, as well as lets you customize the way of how your data will be stored.

The dissertation also describes the prototype of FrameTrail, as well as case studies designed to validate him and how they were directed to use the most of the functionalities provided by the framework.

Keywords: Trail, Context Aware, Framework.

Lista de Figuras

Lista de Tabelas

Lista de Siglas

GIS Geographic Information System

GPS Global Positioning System

GSM Global System for Mobile Communication

HTTP Hypertext Transfer Protocol

IDE Integrated Development Environment

P2P Peer to peer

PC Personal Computer

PDA Personal Digital Assistant

URI Uniform Resource Identifier

XML Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

A evolução da tecnologia tornou possível a redução dos preços dos dispositivos móveis, principalmente telefones celulares e computadores portáteis, tornando-os cada vez mais acessíveis à população. Além disso, com a disseminação do acesso a internet através de redes sem fio *WiFi* (IEEE, 2007) e 3G (IMT, 2000), o uso de sistemas que suportam a Computação Móvel (Satyanarayanan, 1996) vem se tornando cada vez mais frequente. Como uma consequência natural a esse fato, os estudos voltados aos sistemas de localização (Hightower, 2001) sofreram avanços, fazendo com que fosse possível usar essas informações para prover serviços regionalizados (Serviços Baseados em Localização (MobileIN, 2001)).

Sendo assim, o termo Computação Sensível ao Contexto (Dey, 1999), nasceu da união da localização do usuário com os serviços que consideram suas informações de contexto. Esse novo conceito se refere a sistemas computacionais que possibilitam a interação do usuário com o ambiente ao seu redor. Com a disseminação da computação sensível ao contexto, uma necessidade que se tornou evidente foi o registro e o acompanhamento da movimentação do usuário durante a execução de suas atividades. Para suprir essa necessidade, foram desenvolvidas diversas pesquisas com a finalidade de definir como representar o histórico dos contextos visitados por um usuário durante um período de tempo (Smith, 2008). Essas pesquisas resultaram na criação de um conceito denominado Trilha ((Levene, 2002), (Driver, 2007)), que consiste no registro dos contextos onde uma determinada entidade passou, assim como, as atividades em que essa entidade estava envolvida durante aquele momento.

Apesar de haver um consenso no significado e na importância das trilhas quando estamos trabalhando com sistemas móveis, ainda não há uma maneira única de representar esses dados. Essa deficiência faz com que cada aplicativo que precise gerenciar informações referentes às trilhas tenha que desenvolver todos os componentes necessários. Esse tipo de desenvolvimento muitas vezes inviabiliza o projeto, visto que o uso de trilhas normalmente é tido como um requisito do sistema e não como sua atividade fim, fazendo com que investimentos pesados no seu desenvolvimento sejam desaconselháveis.

Além da duplicação de código, a falta de uma ferramenta que possibilite o gerenciamento de trilhas faz com que seja praticamente inviável prover a integração entre duas aplicações distintas. Isso ocorre visto que o processo de conversão de dados tende a ser custoso, o que torna sua adoção impraticável. Outro problema é quanto à mão-de-obra qualificada, pois como cada aplicação trabalha com trilhas do seu modo, um desenvolvedor não consegue aproveitar seu conhecimento adquirido em projetos passados devido às diferenças no tratamento dessas informações.

1.2 Definição do Problema

Baseado no estudo realizado sobre *frameworks* para o desenvolvimento de aplicações orientadas a trilhas, que analisou os seguintes projetos Hermes (Driver, 2007), TrailTRECer (Gams, 2002), Startrack (Ananthanarayanan, 2009), Nidaros (Wang, 2005) e Ubitrail (Silva, 2009), pode-se afirmar que esses projetos apresentam restrições. Essas restrições acabam limitando o funcionamento das aplicações desenvolvidas a partir deles. As restrições identificadas são:

1. Restrição de domínio: Alguns projetos tratam de um tipo de aplicação específica, ou seja, não podem ser estendidos para atender outras áreas de interesse;

2. Infraestrutura limitada: Os projetos que trabalham com trilhas suportam o uso de dispositivos móveis, mas em geral eles necessitam de um ou mais servidores para fazer a integração dos dados.

Dessas duas limitações, a mais crítica é a limitação referente à infraestrutura, pois nem sempre será possível acessar um servidor específico quando estamos tratando de aplicações móveis, sendo assim surge a seguinte questão de pesquisa:

- Como desenvolver aplicações orientadas a trilhas de maneira ágil, independente de domínio e sem limitações quanto a infraestrutura?

1.3 Objetivos

Esse trabalho propõe um *framework* para o desenvolvimento de aplicações com suporte à sensibilidade do contexto e ao gerenciamento de trilhas, chamado FrameTrail. Esse *framework* define uma abstração de dados para trabalhar com trilhas, assim como, define uma metodologia padrão de acesso a essas informações. Outro aspecto importante desse *framework* é quanto a sua necessidade de ser multiplataforma, pois as informações referentes ao contexto do usuário podem ser detectadas pelos mais diferentes tipos de dispositivo. Por mais que o uso de trilhas possa ter diferentes tipos de aplicação, todas elas compartilham uma série de operações básicas.

Sendo assim, no momento em que uma ou mais aplicações usam uma mesma abstração de conceitos, a integração entre elas se torna mais simples e fácil, pois não é necessário fazer nenhum tipo de conversão de dados. Outra vantagem no uso de um *framework* é que ele se propõe a definir uma maneira padrão de trabalho que tem por objetivo evitar que cada projeto desenvolvido nessa área necessite implementar as suas funcionalidades básicas.

Para atender essas características, podemos destacar os seguintes objetivos específicos:

- Definir um *framework* para o desenvolvimento de aplicações com suporte a trilhas que seja de propósito geral;
- Definir um *framework* capaz de ser executado por uma grande variedade de dispositivos, móveis ou não, mas que funcionem fazendo o uso do mesmo processo de execução e que se integrem de maneira transparente;
- Modelar o *framework* para que a sua extensão seja fácil, com a possibilidade de introdução de novos mecanismos para a detecção do contexto, comunicação e armazenamento de dados;
- Implementar o *framework* com tecnologias de fácil aceitação, que permita a reusabilidade e que possa ser executado em diversos ambientes;
- Avaliar o *framework* através de aplicações que façam uso dele;
- Avaliar a utilização do *framework* durante o processo de desenvolvimento, identificando os benefícios trazidos pelo seu uso.

1.4 Metodologia e Organização do Trabalho

A organização da dissertação reflete a metodologia usada na elaboração do trabalho. Inicialmente foram feitas pesquisas a respeito dos conceitos necessários para o desenvolvimento de frameworks orientados a trilhas. Estas pesquisas estão descritas no capítulo 2. A segunda etapa encontra-se no capítulo 3 e contempla um estudo exploratório sobre os trabalhos já desenvolvidos e relevantes a essa área, assim como uma comparação entre eles.

Baseado nos estudos feitos nos capítulos 2 e 3, ficou evidente que existe uma carência quando se fala de *frameworks* para o desenvolvimento de aplicações orientadas a trilhas. Os trabalhos estudados apresentam limitações quanto a sua arquitetura, pois só trabalham com ambientes cliente-servidor, além de não definirem um conceito padrão para o tratamento das trilhas. Motivado por essas limitações, o capítulo 4 apresenta uma *framework* denominado FrameTrail. Nesse capítulo estão

descritos os conceitos definidos para trabalhar com trilhas, os requisitos identificados para atender as funcionalidades básicas, assim como o modelo de arquitetura elaborada para suprir as limitações supracitadas.

O capítulo 5 traz os aspectos de implementação que foram seguidos no desenvolvimento desse trabalho, enquanto o capítulo 6 trata da avaliação do mesmo. Por último, o capítulo 7 discorre sobre as principais contribuições desse trabalho;

2 CONCEITOS BÁSICOS

Esse capítulo apresenta os conceitos básicos para o entendimento desse trabalho. Primeiramente são tratados os conceitos referentes à computação sensível ao contexto. Depois é apresentado o conceito de trilhas e por último, é apresentado o conceito de *framework*.

2.1 Computação Sensível ao Contexto

A mobilidade do usuário é de extrema importância na execução das suas tarefas, pois a cada novo ambiente em que ele se encontra novas informações e desafios são apresentados. Com base nisso, e aliado a grande disseminação das redes sem-fio, o modelo computacional chamado "Computação Sensível ao Contexto" está cada vez mais popular no desenvolvimento de aplicações, tanto no meio cientifico quanto na indústria. Segundo (Dey, 1999), o contexto caracteriza a situação de uma entidade, que pode ser uma pessoa, um lugar ou um objeto. Os contextos e suas informações podem ser originários de diversas fontes de dados, como por exemplo, sensores de localização, sensores de temperatura e monitores de redes de computadores.

Através da sensibilidade do contexto é possível relacionar informações explicitas referentes ao usuário como, por exemplo, suas preferências e objetivos, com informações implícitas inerentes a uma situação em particular. Assim é possível que uma aplicação indique para o usuário de maneira pró-ativa que um determinado evento de seu interesse está programado para acontecer na região onde ele se encontra. Para (Satyanarayanan, 1996), mobilidade exige adaptabilidade, ou seja, os elementos relacionados à localização do usuário, tanto recursos disponíveis no mundo

real, quanto usuários próximos a ele, devem ser considerados pelos sistemas computacionais.

Considerando aplicações sensíveis ao contexto, um dos aspectos mais importantes é como esses sistemas são capazes de identificar, adquirir e tratar os elementos que representam um contexto. As informações relevantes de um contexto podem ser classificadas em cinco categorias: quem, o quê, onde, quando e por quê (Abowd, 2000). Sendo assim, a computação sensível ao contexto se beneficia do uso de informações contextuais para aprimorar a interação com seus usuários. O contexto pode descrever informações sobre localização, dispositivos, perfis de equipamentos e da rede, atividades, objetos computacionais e outros (Chen, 2000). Apesar da complexidade do gerenciamento do contexto, já podemos ver algumas soluções propostas para infraestruturas de software para suporte e execução de aplicações sensíveis ao contexto (*context-aware applications*) em trabalhos como (Chen, 2002), (Ranganathan, 2003), (Henricksen, 2004), (Sacramento, 2004).

2.2 Trilhas

O conceito de trilhas é amplo e pode ser empregado de diversas maneiras. Uma dessas maneiras foi apresentada por Vannevar Bush (Bush, 1945) através de uma máquina chamada *Memex* (*Memory Extension*). Essa máquina era projetada para auxiliar o homem no armazenamento de grandes volumes de informação. Para Bush, a memória humana é definida como um conjunto de associações de idéias. Sendo assim, quando nos deparamos com a definição de algum conceito, imediatamente nosso cérebro nos remete as idéias associadas a ele. Esse processo foi definido como a Trilha dentro do *Memex*. Resumindo, quando um usuário do *Memex* estivesse pesquisando algum assunto específico e durante esse processo de pesquisa ele acessasse outro assunto qualquer, ambos eram relacionados e seu conteúdo era registrado em uma trilha pessoal.

Outra proposta foi apresentada por (Levene, 2002) e se baseava na idéia de que a memória humana e o processo de aprendizagem aconteciam através da

reestruturação e filtragem de experiências. Nesse trabalho, a trilha era representada como uma sequência de informações interconectadas geradas a partir do processo de aprendizagem. Essas informações eram gravadas através do *experience recorder*, que era uma maquina projetada para gravar as tarefas cotidianas das pessoas, através de experiências particulares.

Com a evolução da computação sensível ao contexto tornou-se possível a manipulação de trilhas levando em consideração as características do ambiente onde o usuário se encontra. Com base nisso, trabalhos como o apresentado por (Driver, 2007) foram possíveis de serem desenvolvidos. Nesse trabalho, a trilha tinha como foco o gerenciamento das atividades desempenhadas pelo usuário levando em consideração o seu contexto.

Outro estágio dessa crescente evolução foi apresentado pelo UbiTrail (Silva, 2009), onde os aspectos da computação ubíqua são levados em consideração para a definição da trilha de uma entidade. É importante ressaltar que nesse trabalho a trilha não está relacionada apenas a pessoas e sim a entidades, onde uma entidade pode representar qualquer coisa que se deseja criar ou gerenciar trilhas, sejam elas pessoas, usuários ou qualquer tipo de recurso computacional (computadores, *notebooks* e *smartphones*). Outro aspecto importante desse trabalho é o fato de que cada entidade possuí uma trilha única em todo o ambiente do UbiTrail. Essa trilha é representada pela coleção de contextos visitados por uma determinada entidade.

2.3 Framework

Um *framework* pode ser definido como uma estrutura semi-acabada de software utilizada para o desenvolvimento de aplicações (Crespo, 2000). Segundo Fontoura, o framework deve ser preenchido através da instanciação de seus componentes, possibilitando assim a criação de novas aplicações que possam interagir com o seu domínio-alvo (Fontoura, 1999).

Para que um *framework* possa ser instanciado, ele deve definir uma arquitetura que ofereça construtores básicos para tal. Além disso, deve definir também *hotpots,* que são os pontos de extensão do *framework*, para que as aplicações desenvolvidas a partir dele possam criar suas funcionalidades específicas. ((Pree, 2000), (Schmidt, 2001)).

Uma característica que diferencia os *frameworks* das bibliotecas comuns utilizadas para o desenvolvimento de software é a sua capacidade de inversão de controle (Crespo, 2000). Ou seja, enquanto as bibliotecas disponibilizam componentes e funções que devem ser invocadas pela aplicação fim, o framework oferece mecanismos para que seja possível agregar novas funcionalidades através do acoplamento de componentes que serão executados durante o seu funcionamento (Fayad, 1999).

Os *frameworks* podem ser classificados sob diferentes aspectos. Um aspecto muito comum usado para isso é quanto ao domínio do problema, ou seja, se o *framework* é de aplicação, de domínio ou de suporte. Outro aspecto importante é quanto a sua arquitetura, que pode ser definia entre as seguintes: em camadas, *pipes and filters*, MVC, PAC, reflexiva, microkernel, *blackboard* e *broker*. Além disso, eles podem ser classificados quanto ao seu uso, seja ele *White-box* ou *Black-box*. (Crespo, 2000).

Existem diversas metodologias para o desenvolvimento de *frameworks* (Mattsson, 1996). Aqui, citaremos três processos de desenvolvimento de *frameworks* orientados a objetos.

1. **Processo baseado em experiências de aplicações já desenvolvidas.** Esta metodologia parte do princípio que já foram desenvolvidas algumas aplicações no domínio do problema (foco do *framework*). Assim, em função do desenvolvimento já realizado, procura-se identificar características comuns às aplicações de modo a abordá-las em um *framework*. A experiência ganha com a utilização do *framework* em aplicações posteriores ao desenvolvimento do *framework* serve como um aperfeiçoamento do próprio *framework*.

2. **Processo baseado na análise do domínio.** Nesta metodologia, o primeiro passo é a análise do problema proposto, que procura identificar abstrações do domínio. Nesta etapa também são estudadas aplicações já existentes, que tem o mesmo escopo do problema (embora, isto nem sempre seja possível, pois as aplicações deste domínio podem ainda não existir). A partir das observações, o *framework* é construído, juntamente com uma aplicação teste a fim de avaliar o desenvolvimento do *framework* e refiná-lo.

3. **Processo de desenvolvimento utilizando *design patterns*.** O primeiro passo deste processo é a construção de uma aplicação no domínio. A partir deste ponto, são identificados na aplicação *design patterns*, que serão aplicados na construção do *framework*.

Em função das características de um framework, podemos observar pelo menos três benefícios do uso de frameworks: modularidade, reutilização e extensibilidade (Crespo, 2000).

2.4 Considerações sobre o Capítulo

Este capítulo apresentou os conceitos de computação sensível ao contexto, trilhas e *framework*. Esses conceitos direcionam este trabalho que define um *framework* para o desenvolvimento de aplicações orientadas a trilhas. A sensibilidade de contexto será explorada para caracterizar cada etapa da trilha, detalhando a situação onde ela ocorreu. O conceito de trilha que será adotado por esse trabalho é o mesmo introduzido pelo projeto UbiTrail, pois este modelo tem como principal vantagem a possibilidade de atender qualquer tipo de entidade.

Quanto às características do *framework*, podemos dizer que, sob seu aspecto de domínio, ele é um *framework* de aplicação, pois visa prover funcionalidades específicas para o gerenciamento de trilhas. Sob a perspectiva de arquitetura pode-se dizer que o trabalho proposto está organizado em uma arquitetura reflexiva, pois seu funcionamento está direcionado ao registro e invocação de componentes a partir do

seu fluxo de trabalho. E quanto ao seu uso, o *framework* pode ser classificado nas duas categorias (*white-box* e *black-box*), pois possui componentes que devem ser estendidos pelo desenvolvedor durante a sua instanciação, além de ter a possibilidade de apenas agregar funcionalidades a outras aplicações através do uso de seus componentes já disponíveis.

No próximo capítulo serão apresentados os trabalhos relacionados a essa proposta, analisando suas principais características e comparando elas com o *framework* proposto.

3 TRABALHOS RELACIONADOS

Neste capítulo serão apresentados os trabalhos relacionados com o tema de estudo desta dissertação. As pesquisas realizadas tiveram como objetivo identificar quais eram os trabalhos já realizados sobre *frameworks* para o desenvolvimento de aplicações com suporte a trilhas.

3.1 Hermes

Hermes é um *framework* para o desenvolvimento de aplicações com suporte a trilhas. O *framework* está organizado em série de componentes e serviços que podem ser utilizados por aplicações para tratamento de trilhas. Dentro do conceito do Hermes, trilha é uma coleção de atividades programadas de forma contextual, ou seja, a ordem de execução dessas atividades é definida levando em consideração o contexto atual do usuário. Essas atividades são representadas por um modelo genérico que pode ser usado para satisfazer os requisitos de gerenciamento de uma grande variedade de aplicações (Driver, 2007). A Figura 1 mostra a arquitetura do projeto Hermes, através do seu diagrama de componentes.

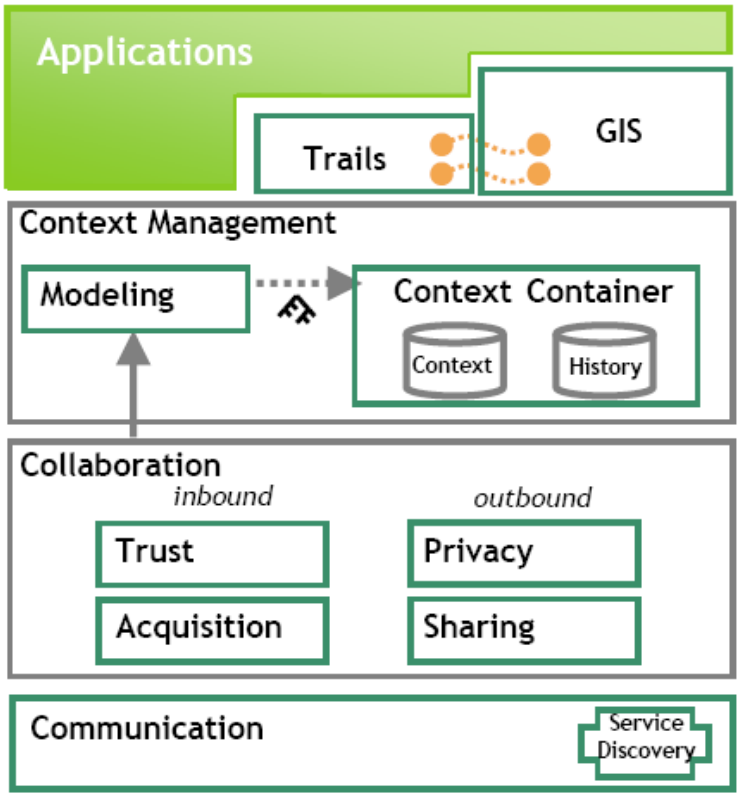

Figura 1. Arquitetura do framework Hermes (Driver, 2007).

A estrutura do projeto Hermes é separada em camadas, onde cada camada é isolada e independente das demais, ou seja, alterações feitas no funcionamento interno em uma camada não devem representar impactos nas demais camadas da estrutura do *framework*. A abordagem utilizada para o desenvolvimento dessas camadas foi a *bottom-up*, ou seja, cada camada é construída a partir das funcionalidades das camadas mais abaixo. Sendo assim, as camadas são as seguintes:

1. ***Communication*:** É a camada responsável pela integração do *framework* com sensores externos, com a infraestrutura da aplicação, assim como com outros dispositivos computacionais através de vários tipos de redes diferentes;

2. ***Collaboration*:** São os componentes responsáveis pela colaboração entre as aplicações desenvolvidas pelo *framework*. Esses componentes estão separados em duas colunas, os itens à esquerda tratam os aspectos de requisição e

confiança entre as aplicações enquanto os componentes à direita tratam os aspectos referentes ao compartilhamento e privacidade das informações;

3. ***Context Management*****:** É responsável pela modelagem e gerenciamento dos contextos. A modelagem é o processo de conversão dos dados do ambiente para a representação de contexto definida pelo Hermes. O gerenciamento é responsável pelo armazenamento do contexto, que pode ser tanto em memória quanto em disco, dependendo do objetivo de cada aplicação;

4. ***Applications*****:** Essa última camada representa as aplicações desenvolvidas a partir do Hermes. Para facilitar o desenvolvimento dessas aplicações, o Hermes disponibiliza dois componentes externos que podem ser usados pelas aplicações para o tratamento das informações gerenciadas pelo *framework*. Esses componentes são: o *GIS (Geographic Information System)* que é responsável por manter a representação do mundo real estruturado geograficamente e o *Trails* que é responsável pelo gerenciamento das trilhas utilizadas pelas aplicações.

O conceito de trilha adotado pelo projeto Hermes define que trilha é uma coleção de atividades programadas de forma contextual, ou seja, seu foco é voltado para o gerenciamento de atividades. Sendo assim, suas funcionalidades são direcionadas para a geração automática da ordem de execução dessas atividades. Essa ordenação é feita levando em consideração informações de contexto.

Outra característica importante do projeto Hermes é o seu suporte a reconfiguração dinâmica de trilhas, ou seja, de acordo com o contexto e o andamento de suas atividades a trilha em questão é reavaliada e a ordem de execução das suas atividades pode ser alterada.

Além das funcionalidades referentes à geração e adaptação das trilhas, o projeto Hermes também disponibiliza componentes utilitários que servem para fazer o armazenamento dos dados. Esses componentes apenas gravam e recuperam os dados das trilhas a partir dos arquivos pré-definidos. Como não existe um repositório de

trilhas no projeto, fica a cargo de cada aplicação determinar o local onde os dados serão salvos.

3.2 *TrailTRECer*

O projeto *TrailTRECer* (Gams, 2002) consiste em um *framework* para o rastreamento de informações digitais, auxiliando na navegação e na busca de conteúdo. Para esse projeto, o termo trilha é definido como o conjunto de documentos digitais que um usuário acessou ao executar uma determinada atividade. Esse *framework* trata apenas de trilhas baseadas em *software*, ou seja, apenas da interação do usuário com qualquer tipo de informação proveniente de um documento digital, tal como um arquivo texto, planilha de dados, página de internet, *email* entre outros.

O processo de trabalho desse projeto consiste na coleta de informações provenientes de qualquer tipo de *software* para a composição da trilha. Portanto, para prover essa integração, o modelo do *TrailTRECer* trata as trilhas como objetos através da conversão dos dados dos aplicativos específicos em uma estrutura de dados padronizada. Nessa estrutura uma trilha é representada como uma coleção de *TrailMarks*. Qualquer documento que possa ser representada por uma URI pode ser convertida em um *TrailMark* e assim fazer parte a trilha.

A arquitetura do *TrailTRECer* é baseada em agentes e dividida em duas partes, a *Group Platform*, que trata os componentes que agregam as informações coletadas pelos agentes e a *User Platform* que é responsável por rodar os agentes e integrá-los com os aplicativos que podem gerar informações para a criação dos *TrailMarks*. A Figura 2 ilustra os componentes da arquitetura do *TrailTRECer.*

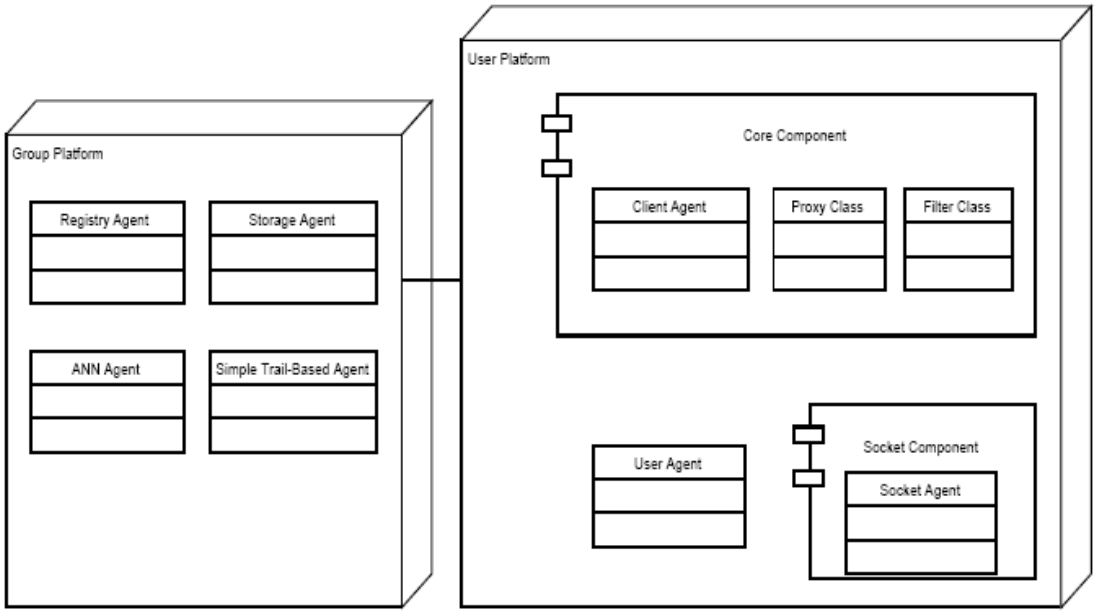

Figura 2. Arquitetura do *TrailTRECer* (Gams, 2002).

Como o foco de trabalho do projeto *TrailTRECer* é voltado para a geração e visualização de trilhas, ele não possui um repositório especifico onde as trilhas podem ser consultadas por diferentes tipos de critérios. O processo de leitura e gravação dos dados da trilha é feito pelo *Storage Agent* que armazena esses dados em arquivos próprios do *framework*.

3.3 Startrack

O *framework StarTrack* (Ananthanarayanan, 2009) é um sistema que prove uma série de operações para a manipulação de trilhas. O conceito de trilha usado por esse *framework* indica que cada trilha representa uma rota contínua baseada em eventos discretos, ou seja, dispositivos móveis coletam os dados da trilha e os enviam para o servidor quando possível. *Startrack* inclui funções para armazenar, comparar, agrupar, indexar e consultar trilhas. Através dessas características, o *framework* se propõe a servir de base para a construção de serviços baseados em trilhas e disponibilizar o seu acesso em larga escala.

Dentre todas as funcionalidades do *Startrack*, podemos ressaltar as seguintes contribuições:

1. a representação de uma abstração padronizada das trilhas, ou seja, cada aplicação pode definir a quais serão os dados salvos em cada etapa da trilha;
2. a criação de algoritmos otimizados para trabalhar com trilhas, principalmente as operações de comparação e agregação;
3. possibilita a manipulação de grandes volumes de dados, através do cruzamento dos dados de trilhas de vários usuários.

Cada trilha dentro do *StarTrack* é uma coleção de lugares visitados por um dispositivo, ordenados no tempo. O principal objetivo de cada dispositivo é ser a representação virtual do usuário que o carrega. Cada um dos registros que compõe a trilha é chamado de *Track Entry*. As trilhas são geradas através das aplicações específicas e são salvas em uma base de dados central, juntamente com as trilhas gravadas previamente por outros usuários.

Uma trilha pertence a um determinado usuário e o mesmo define quem tem acesso a ela. A *Track Entry* consiste em um registro com localização, data, hora e pode conter algum tipo de informação específica de cada aplicação. Para prover uma melhor flexibilidade sobre essas informações específicas, o *StarTrack* armazena as trilhas no formato XML, o que possibilita que cada aplicação desenvolva o *layout* das informações que pretende armazenar em cada trilha.

O *StartTrack* é baseado na arquitetura cliente-servidor, conforme mostra a Figura 3. A figura mostra um exemplo do seu funcionamento onde existem três tipos de dispositivos envolvidos, um telefone celular, um computador pessoal e um servidor.

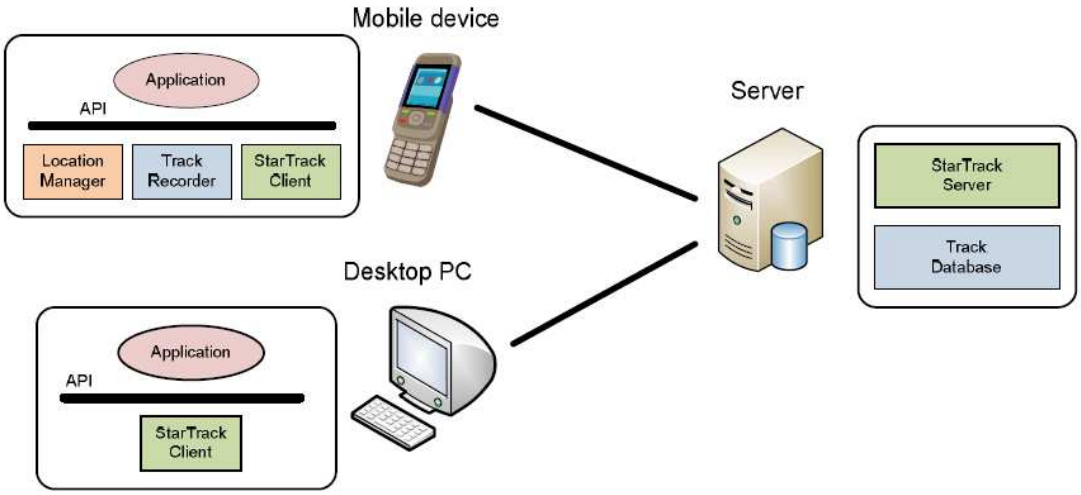

Figura 3. Arquitetura do StarTrack (Ananthanarayanan, 2009)

Para o funcionamento do *StarTrack* é necessário que o dispositivo móvel em questão possua a capacidade de determinar sua própria localização. É através do *Location Manager* que o framework gerencia a posição de cada dispositivo, tornando-o assim compatível com diversas tecnologias de posicionamento existentes, tais como *GPS*, *GSM localization*, triangulação de antenas entre outros ((Paramvir, 2000), (LaMarca, 2005), (Veljo, 2005)).

Outro componente importante nos dispositivos móveis é o *Track Recorder,* pois é através dele que o *framework* cria uma nova *Track Entry* de tempos em tempos para ir gravando a trilha do dispositivo em questão. Por padrão as trilhas são armazenadas localmente deixando a cargo da aplicação específica determinar qual o melhor momento de salvar essas trilhas no servidor. Cada aplicação também tem a liberdade de definir quando salvar uma *Track Entry,* assim como, determinar quando uma trilha nova inicia ou acaba.

O *StarTrack* tem também o foco de prover funcionalidades de manipulação e gerenciamento de trilhas para aplicativos que rodem em outros tipos de dispositivos, tais como, os computadores pessoais (*desktops*). Os aplicativos feitos a partir do pacote *StarTrack Client* podem acessar os dados de qualquer servidor de trilhas do StarTrack (desde que tenham permissão para esse acesso), podendo assim executar

operações de consulta nesses dados. Um exemplo desse desenvolvimento é uma aplicação para a sugestão de caronas entre colegas de trabalho. Cada trabalhador possui em seu telefone celular uma aplicação que grava o caminho que ele faz de casa até o trabalho e armazena esses dados no servidor do *StarTrack*. A aplicação que executa no *desktop* seleciona grupos de colegas que fazem caminhos parecidos no mesmo horário, e sugere visualmente através de um mapa o caminho em comum para eles.

3.4 Nidaros

A proposta do *framework* Nidaros (Wang, 2005) é possibilitar o desenvolvimento ágil de aplicações conscientes de localização para disponibilizar ao usuário conteúdo multimídia localizado. O Nidaros propõe um processo de desenvolvimento de *software* que prevê a definição de interfaces gráficas de maneira padronizada com suporte a diferentes tipos de dispositivos móveis, respeitando suas principais características e restrições.

Uma das limitações apresentadas nos sistema sensíveis à localização é a de usar apenas um tipo de tecnologia de posicionamento. Para solucionar esse problema, o *framework* Nidaros faz uso de um servidor de localização que une as principais tecnologias de posicionamento, tais como, *GPS*, *GSM*, *Bluetooth*, Infravermelho e triangulação de antenas e agrega essas informações em uma representação única de dados de localização. O componente de *Runtime* do *framework* se encarrega de se comunicar com esse servidor e retornar para o cliente a sua posição mais aproximada.

A Figura 4 mostra uma visão geral do Nidaros *Framework* durante o processo de desenvolvimento de aplicações conscientes de localização. O framework abrange as etapas de criação e execução desse tipo de sistema.

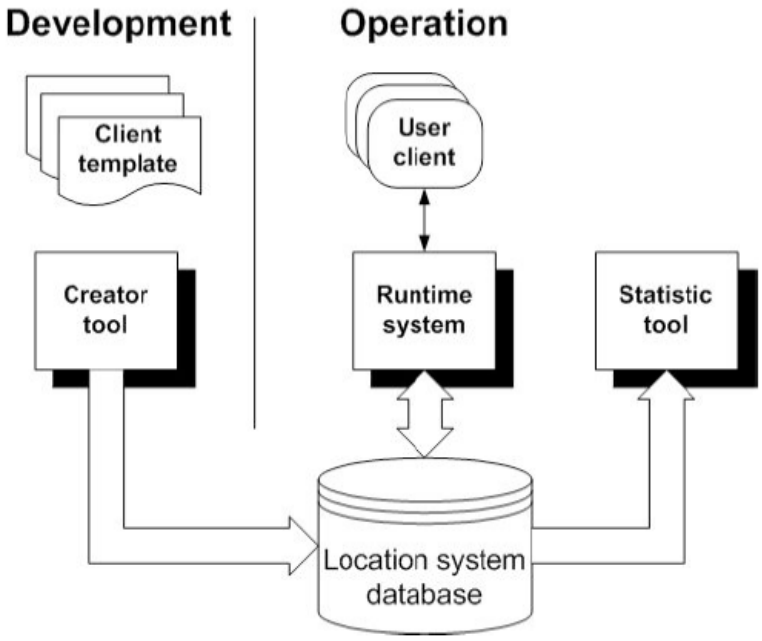

Figura 4. Processo de desenvolvimento de aplicações com o Nidaros Framework (Wang, 2005).

A fase de desenvolvimento é feita através do uso da ferramenta de criação que é responsável pelo mapeamento de conteúdos localizados. Para a interface da aplicação, o sistema faz o uso do conceito de *templates*, baseados em *XML*, que geram os componentes de tela de acordo com as limitações do dispositivo onde o cliente está sendo executado.

Os componentes *runtime system* e *statistic tool* são responsáveis pela execução do sistema, enquanto a base de dados e o *location server* executam as tarefas necessárias a integração do ambiente como um todo. A Figura 5 mostra o diagrama físico do *runtime system* e como seus componentes internos se relacionam.

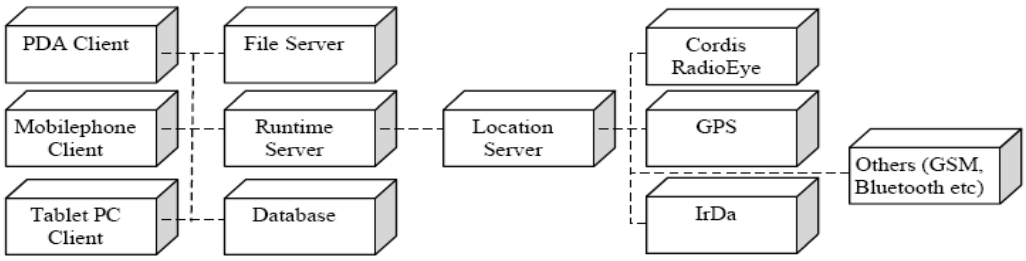

Figura 5. Componentes do *Runtime System* do Nidaros *Framework* (Wang, 2005).

O diagrama começa expondo os três tipos de clientes diferentes usados para validar o framework, são eles: um *PDA*, um telefone celular e um *tablet pc*. Esses componentes se comunicam através de uma rede sem fio com o servidor. Os quatro principais componentes do *Runtime System* do Nidaros são:

1. ***File Server*:** é onde os arquivos multimídia são armazenados e acessados pelos clientes móveis. Nem todos os dispositivos são capazes de armazenar grandes quantidades de dados localmente, para esses casos o servidor os envia sob demanda. Os dispositivos com capacidade de armazenamento recebem os arquivos e os armazenam localmente para agilizar o acesso feito pelo usuário;
2. ***Runtime Server*:** é responsável por tratar as chamadas originadas dos clientes e executar os serviços requisitados.
3. ***Location System Database*:** armazena todas as informações que são usadas pelos clientes e pelo servidor. Essa base de dados também é usada pelo *creator tool* no desenvolvimento das aplicações e pela *statistics tool* para analisar e identificar os padrões de deslocamento dos usuários;
4. ***Location Server*:** gerencia o posicionamento dos clientes fazendo o uso de um ou mais tipos de tecnologias de localização diferentes.

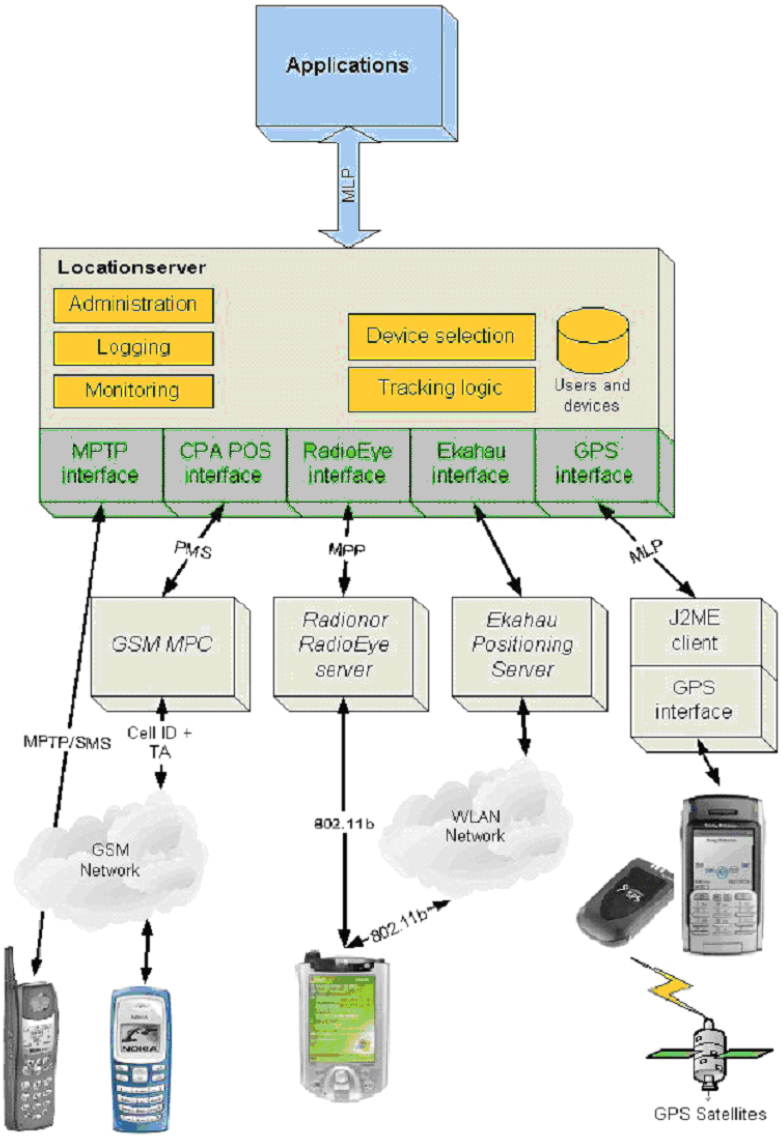

Figura 6. Componentes do *Location Server* (Wang, 2005).

A Figura 6 mostra os componentes internos do servidor de localização e como eles se relacionam com os dispositivos móveis que os acessam. Um aspecto importante desse componente, é que ele mesmo se encarrega de converter as métricas de cada sensor em coordenadas padrões, fazendo o uso de um sistema único de coordenadas.

Dentro do conceito proposto pelo Nidaros, a trilha é o conjunto de registros do que indicam os lugares onde cada dispositivo teve acesso. Esses registros ficam armazenados na base de dados de localização do *framework*. Essa base de dados é

alimentada pelo *Runtime System* à medida que o *Location Server* informa a localização atual do dispositivo para aplicação cliente. As ferramentas fornecidas pelo *framework* estão voltadas a gerar estatísticas sobre essa base de dados.

3.5 Ubitrail

O UbiTrail (Silva, 2009) é um modelo que se propõe a gerenciar informações sobre a trilha de uma entidade. Dentro dessa proposta, a trilha é uma representação digital do histórico dos contextos visitados por uma entidade. A entidade é um dispositivo computacional que pode representar um usuário específico, ou seja, a pessoa que faz uso dele, como também pode ser qualquer tipo de dispositivo na qual se deseja criar e gerenciar trilhas. Outro aspecto importante do UbiTrail é o seu suporte a consciência de contexto, ou seja, não apenas a localização da entidade é considerada, mas também o que ela está fazendo e o ambiente ao seu redor (Silva, 2009).

Dentro do conceito do UbiTrail, as informações relacionadas com o deslocamento de uma entidade são organizadas em uma trilha única e contínua. Essa trilha é composta por uma sequência de registros chamados *ptrail* (*piece trail*). Cada *ptrail* é constituído pelos seguintes atributos: entidade, recurso, evento, extensão e localização. Os valores dos atributos entidade e recurso são mantidos de forma estática dentro da *ptrail* e os demais são atualizados dinamicamente de acordo com a ação e o deslocamento da entidade. O *trailpoint* é o processo que adiciona os dados de uma *ptrail* dentro da trilha da entidade e pode ocorrer pela solicitação da entidade que está usando o dispositivo móvel, assim como quando a aplicação identifica automaticamente um novo evento como, por exemplo, a entrada e saída de um contexto.

A arquitetura do UbiTrail é composta por três tipos de componentes, os provedores externos, o *UbiTrailServer* e o *UbiTrailClient*. Os dois provedores externos considerados na arquitetura do UbiTrail são os provedores de contexto e o de localização, o que o torna independente de tecnologia nesses dois quesitos. O

UbiTrailServer é responsável por gerenciar a base de trilhas e executar os serviços de acesso a esses dados, enquanto o *UbiTrailClient* é executado pelos dispositivos móveis para possibilitar a comunicação com o *UbiTrailServer*.

O *UbiTrailServer* é composto por três camadas e três repositórios de dados, conforme mostra a Figura 7 (a). A camada *TrailServices* é responsável por executar os serviços de acesso as trilhas, que podem ser os serviços básicos (providos pelo ambiente) e os especializados (disponibilizados pelas aplicações específicas). A *TrailContext* gerencia integração com os provedores externos, enquanto a última camada é organizada em três módulos:

1. *TrailManager*: responsável por gerenciar a comunicação entre os componentes do *UbiTrailServer* com o *UbiTrailClient*;
2. *TrailRegister*: autentica e manipula as entidades envolvidas durante a execução da aplicação;
3. *TrailRepository*: é a interface de acesso para as informações dos repositórios de dados.

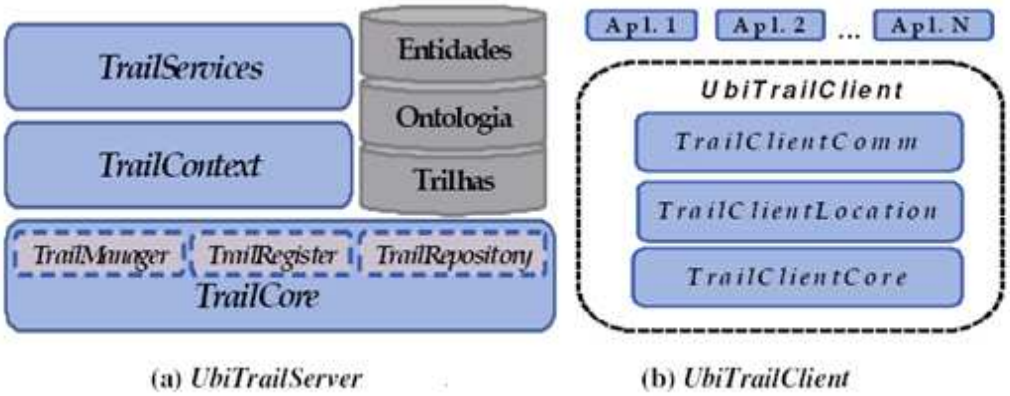

Figura 7. Arquitetura do UbiTrail (Silva, 2009).

Uma vez que o *UbiTrailServer* está funcionando, ele em si não executa nenhum tipo de aplicação, o responsável por isso é o *UbiTrailClient* que executa no dispositivo móvel. A Figura 7 (b) mostra os componentes do *UbiTrailClient*. Ele é organizado em três camadas e cada uma delas possui um objetivo bem especifico. A primeira camada,

a *TrailClientComm*, é responsável por enviar requisições para o *UbiTrailServer* e tratar suas respostas. A *TrailClientLocation* é a camada que trata da localização do usuário, através da sua integração com os provedores externos. Por último está a *TrailClientCore* que integra esses componentes com os recursos físicos do dispositivo onde o cliente está sendo executado.

3.6 Comparativo entre Projetos

Os projetos estudados apresentam como objetivo agilizar o processo de desenvolvimento de aplicações com suporte a trilhas. Esses projetos possuem características diferentes, pois cada um possui um foco específico no tratamento das suas trilhas. Para melhorar o entendimento sobre as contribuições e limitações de cada um destes projetos, foi elaborada a Tabela 1 que apresenta um comparativo das características de cada um deles. Os itens analisados foram definidos em função do foco de trabalho do FrameTrail e são os seguintes:

1. **Considera informações de contexto:** Este critério indica se o projeto mapeia alguma informação de contexto para que as aplicações derivadas dele suportem a interação do usuário com o mundo ao seu redor;

2. **Pode ser estendido para outros domínios ou tipos de aplicação:** indica se o projeto é utilizado para algum tipo de aplicação específica ou se ele é independente de domínio de aplicação e pode ser adaptado para trabalhar com qualquer tipo de trilha;

3. **Oferece uma biblioteca destinada ao desenvolvimento de aplicações:** indica se o projeto pode ser integrado em aplicações existentes ou se é um ambiente próprio que necessita de uma implementação específica;

4. **Arquitetura do modelo:** indica qual arquitetura que as aplicações desenvolvidas devem seguir para fazer uso do *framework* em questão. Alguns trabalhos requerem uso de servidores específicos para localização e armazenamento. Dentro desse quesito as arquiteturas avaliadas foram

classificadas como cliente-servidor, ou seja, aquelas que necessitam de uma infra-estrutura mínima de servidores para o seu funcionamento, e peer-to-peer, que seriam os *frameworks* capazes de executar suas aplicações sem ter que ter obrigatoriamente uma estrutura de servidores para isso;

5. **Mantém informações históricas:** este item indica se o projeto tem suporte ao armazenamento de informações históricas referentes ao deslocamento do usuário;

6. **Local para armazenamento dos dados:** este item classifica os projetos de acordo com o local onde eles salvam os seus dados. Como uma das principais atribuições dos sistemas baseados em trilhas é gravar o deslocamento do usuário, os projetos analisados apresentam suporte para aplicações que executam em dispositivos móveis e seus dados podem ser salvos no próprio dispositivo (local) ou em algum servidor externo (remoto);

7. **Oferece suporte a mais de um tipo de tecnologia de localização:** este item identifica os projetos que possibilitam o uso de mais de um tipo de tecnologia para detectar o posicionamento do clientes envolvidos.

	Característica	Hermes	TrailTRECer	Startrack	Nidaros	Ubitrail
1	Considera informações de contexto	Sim	Sim	Não	Sim	Sim
2	Pode ser estendido para outros domínios ou tipos de aplicação	Sim	Não	Sim	Não	Sim
3	Oferece uma biblioteca destinada ao desenvolvimento de aplicações	Sim	Sim	Sim	Sim	Não
4	Arquitetura do modelo	Cliente-Servidor	Cliente-Servidor	Cliente-Servidor	Cliente-Servidor	Cliente-Servidor
5	Mantém informações históricas	Não	Sim	Sim	Sim	Sim
6	Local para armazenamento dos dados	Remoto	Remoto	Remoto e Local	Remoto	Remoto
7	Oferece suporte a mais de um tipo de tecnologia de localização	Sim	Não	Sim	Sim	Sim

Tabela 1. Comparativo entre os projetos estudados

Conforme foi demonstrado no decorrer desse capítulo, os trabalhos apresentados não tratam o conceito de trilhas da mesma maneira. Para o projeto

Hermes, a trilha é uma lista de atividades que devem ser executadas pelo usuário e seu foco está na criação e reconfiguração dinâmica delas. Enquanto isso, o projeto TrailTRECer usa a trilha como uma ferramenta para registrar quais documentos um usuário teve acesso, ou seja, a trilha reflete o trabalho desempenhado por esse usuário. Os projetos Startrack e Nidaros usam o conceito trilha voltados ao registro do deslocamento dos usuários. Enquanto Startrack tem um foco mais amplo, ou seja, ele visa registrar as trilhas de vários usuários em uma base única, deixando a cargo de cada aplicação definir qual o seu enfoque. O projeto Nidaros é voltado especificamente para a distribuição de conteúdo multimídia baseado na localização do usuário. O projeto UbiTrail une os conceitos de trilhas com a computação ubíqua, ou seja, ele usa a trilha como a coleção de contextos visitados por uma entidade, assim como as atividades realizadas por elas.

Dos projetos apresentados, o único que não faz uso das informações de contexto é o Startrack, pois o seu foco está no registro do deslocamento do usuário não levando em consideração as informações do ambiente onde ele se encontra. Para o projeto TrailTRECEr o contexto representa o *software* que gerou cada etapa da trilha, pois o seu objetivo é registrar quais documentos um determinado usuário acessou. Para os projetos Hermes, Nidaros e UbiTrail, o conceito de contexto é mais amplo e representa a situação atual de uma entidade como um todo, não apenas a sua localização geográfica. Cada um deles deixa aberta a possibilidade das aplicações desenvolvidas a partir deles usarem as informações de contexto que acharem mais relevantes, como por exemplo, a localização geográfica, data, hora e informações meteorológicas.

Quanto ao desenvolvimento de novas aplicativos, os projetos TrailTRECer e Nidaros não podem ser estendidos para outros domínios ou tipos de aplicação, ou seja, eles só contemplam respectivamente as funcionalidades necessárias para o desenvolvimento de *softwares* voltados ao registro dos documentos acessados por um usuário, ou a distribuição de conteúdo multimídia localizado. Enquanto isso, os projetos Hermes, Startrack e UbiTrail definem apenas as funcionalidades de suporte ao

processamento de trilhas deixando a cargo de cada nova aplicação definir como elas irão explorar essas trilhas.

Sob o ponto de vista da maneira como os novos aplicativos são desenvolvidos, apenas o UbiTrail não é um *framework*, ou seja, ele é o único que não disponibiliza bibliotecas para o desenvolvimento de novas aplicações. Sendo assim, o UbiTrail é um ambiente para a execução de serviços com suporte a trilhas que agrega novas funcionalidades através da codificação de novos *Web Services* que são integrados ao servidor do UbiTrail existente.

Um aspecto que ficou evidente sobre os projetos analisados foi quanto a sua arquitetura. Todos os projetos apresentam um modelo de trabalho cliente-servidor, ou seja, eles precisam de uma estrutura central para o registro e acesso a informações. O único projeto que apresenta uma alternativa para falta de servidor é o Startrack, mas apenas para a funcionalidade de salvar os dados da trilha, pois no momento em que for necessário gravar os dados de alguma etapa da trilha e o servidor não estiver disponível, ele prevê a possibilidade de armazenar esses dados localmente e enviá-los posteriormente para o servidor. Esses servidores são responsáveis por manter os dados históricos da trilha, com exceção do projeto Hermes, que não trata o histórico de suas trilhas.

Quanto às tecnologias de localização, todos os projetos possuem suporte para o uso de mais de um tipo, exceto o projeto TrailTRECer que não faz uso da localização física do usuário. O projeto Nidaros possui um servidor específico para determinar a localização de cada usuário e o mesmo define esse posicionamento através de um sistema de coordenadas próprio. Enquanto isso, os demais projetos definem essas tecnologias em seus módulos clientes, ou seja, os dispositivos móveis detectam o seu próprio posicionamento e enviam essas informações periodicamente para os servidores.

3.7 Considerações sobre o Capítulo

O capítulo 3 apresentou um conjunto de propostas existentes para o desenvolvimento de aplicações orientadas a trilhas e identificou suas principais características, limitações e contribuições.

O próximo capítulo apresenta o *framework* FrameTrail, uma nova alternativa para esse tipo de desenvolvimento que busca contornar as limitações dos trabalhos estudados.

4 MODELO

Este capítulo apresenta o *framework* FrameTrail que se propõe a ser uma alternativa para o desenvolvimento de aplicações com suporte a trilhas. O *framework* visa atender requisitos levantados em função da análise feita sobre os projetos discutidos no capítulo 3, principalmente as restrições referentes à arquitetura e as tecnologias envolvidas. O capítulo inicia com uma visão geral do funcionamento do *framework*. Após os seus requisitos são detalhados. Por fim, é apresentada a arquitetura do *framework* e como os seus componentes se relacionam.

4.1 Visão Geral

Esse trabalho propõe um framework para auxiliar no desenvolvimento de aplicações que fazem uso do conceito de trilhas. Trilha é a coleção de registros que armazenam as informações referentes aos contextos visitados por uma entidade e as atividades desenvolvidas por ela durante esse processo. O modelo de trilha usado por este trabalho segue as idéias propostas pelo UbiTrail (Silva, 2009), ou seja, cada entidade possui uma trilha única que registra todos os contextos acessados por ela.

Como o foco do FrameTrail é ser um *framework* para a manipulação de trilhas genéricas, a sua camada de abstração não trata de nenhum domínio de trilha específico, ou seja, trata apenas dos dados para a caracterização do ambiente (contexto), das trilhas e dos dispositivos necessários para o seu funcionamento. Outro aspecto importante gerenciado pelo framework é a possibilidade de execução externa de serviços. Esse tipo de execução implica na possibilidade das aplicações desenvolvidas a partir dele poderem delegar a execução de um ou mais serviços a outros dispositivos. A seguir serão descritos os principais conceitos do *framework*:

- **Entidade:** representa uma pessoa, usuário, recurso computacional ou qualquer outro tipo de objeto para a qual deseja-se criar ou gerenciar trilhas;

- **Contexto:** caracteriza a situação atual da entidade no momento da interação com o sistema, ou seja, onde ela está, quais serviços estão disponíveis, quais entidades estão nesse mesmo ambiente, além de qual é o seu propósito;

- **Região Geográfica:** descreve os lugares onde uma entidade pode estar localizada e essas regiões podem ser organizadas hierarquicamente. Sendo assim, elas podem representar localidades em diferentes granularidades, como por exemplo, continentes, países, cidades, bairros e ruas;

- **Localização Geográfica:** representa a localização de uma entidade dentro de uma região geográfica;

- ***Checkpoint:*** Ponto de salvamento da trilha. Relaciona os dados referentes ao contexto atual da entidade. Pode ou não conter dados específicos de cada aplicação, mas fica a cargo de cada uma delas gerenciá-los. Cada aplicação desenvolvida poderá adicionar novos campos à estrutura do *checkpoint* e esse *checkpoint* poderá ser criado através de uma chamada explícita, ou também poderá ser gerado automaticamente através da criação de eventos parametrizáveis;

- ***Evento:*** representa qualquer acontecimento passível de interação que uma aplicação deseja monitorar através do *framework*. Os eventos são cadastrados por tipo e esses tipos definem quais são as condições para que esse evento aconteça. Os tipos de evento podem ser divididos entre aqueles que criam ou não um *checkpoint*. Os eventos que criam *checkpoints* são usados para gravar as etapas de uma trilha, enquanto os que não criam são usados como meio de comunicação entre a aplicação e o *framework*;

- **Trilha:** Representa a lista de contextos visitados por uma determinada entidade. Cada entidade possui uma trilha única e cada contexto visitado é representado por um *checkpoint*.

- **Sensor:** é um recurso computacional usado para monitorar as condições do ambiente. Cada sensor se encarrega de monitorar um tipo de informação especifica, como por exemplo, tempo ou condição meteorológica. Através das informações dos sensores, o *framework* é capaz de caracterizar um contexto, ou ativar alguma restrição que esteja vinculada a ele.

- **Container:** é um tipo de entidade que se propõe a agregar uma ou mais entidades e gerenciar seus eventos de maneira compartilhada. Através desse conceito, as entidades agregadas podem delegar a geração de seus *checkpoints*, assim como o gerenciamento do seu contexto, para o *container*, economizando processamento e otimizando assim seu consumo de energia (no caso de dispositivos móveis).

Com base nesses conceitos, o funcionamento do *framework* ocorre através de um mecanismo para a invocação dinâmica de eventos parametrizáveis. A parametrização desses eventos depende da finalidade de cada sistema, ou seja, diferentes tipos de aplicativos terão a definição de eventos com maior ou menor nível de granularidade. A Figura 8 mostra esse conceito:

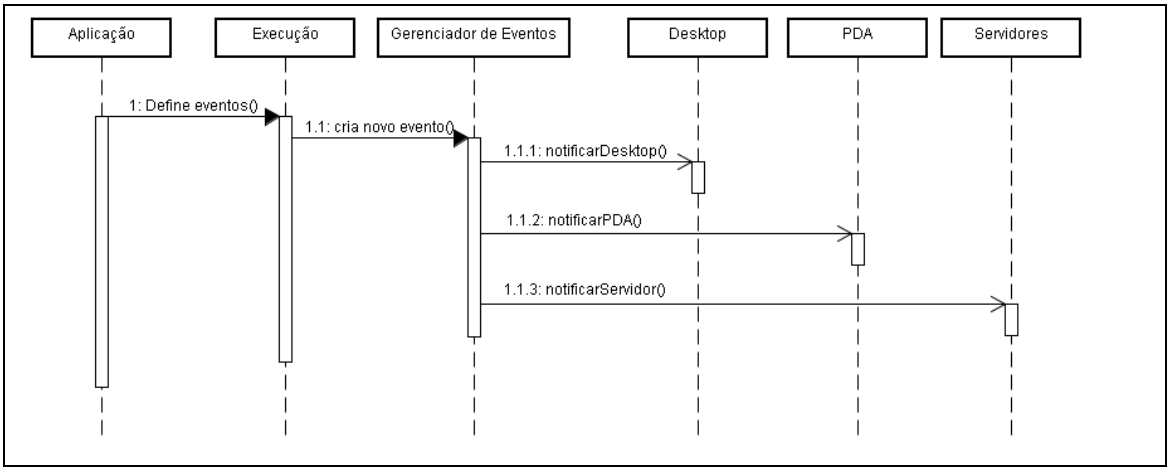

Figura 8. Funcionamento do mecanismo de eventos do *framework*.

Os eventos são definidos pela aplicação podendo ser em tempo de desenvolvimento, através da parametrização das triggers durante a configuração de cada instância do framework, ou em tempo de execução, através da criação dinâmica de *triggers*. Quando a aplicação estiver em execução o *framework* se encarrega de analisar em tempo real o contexto atual e gerar os eventos previamente definidos. Esses eventos podem ser consumidos por uma ou mais aplicações diferentes e essas aplicações podem estar executando em um ou mais dispositivos diferentes. Essa abordagem visa flexibilizar a maneira como cada componente interage com o sistema, pois cada evento pode ser consumido por diversas entidades diferentes e isso torna a extensão do sistema mais simples e fácil, visto que cada tipo de dispositivo terá a capacidade de consumir esses eventos de uma maneira única e integrada.

Cada aplicação que faz uso do FrameTrail registra os sensores que serão responsáveis por monitorar as informações necessárias para o seu funcionamento. Cada sensor deve estar vinculado a algum tipo de *hardware* ou *software* para fazer as leituras das informações pertinentes a ele. É através do conjunto dessas leituras que o *framework* se encarrega de gerenciar os contextos, assim como avaliar as condições das *restrições* configuradas para gerar os eventos vinculados a elas.

Um dos objetivos desse trabalho é se integrar a diversos tipos de dispositivos, entre eles computadores pessoais, telefones celulares, servidores de internet e até mesmo aparelhos de GPS veicular. Essa integração é feita através do uso de provedores externos, que são responsáveis por traduzir os dados processadas entre essas plataformas e o *framework*. Esse processo de tradução é necessário devido às diferentes características dessas plataformas, principalmente relacionado ao seu poder de processamento, capacidade de armazenamento e memória. Sendo assim, o *framework* FrameTrail possui mais de um tipo de implementação, cada uma delas trata as necessidades de um tipo de plataforma, mas todas elas se integram de maneira transparente.

4.2 Requisitos do Framework

Os requisitos funcionais do FrameTrail foram baseados no modelo proposto para atender os objetivos definidos nesse trabalho. Esses requisitos estão representados na Figura 9, que ilustra a relação entre os requisitos identificados com os componentes do *framework*. Para atender todos os requisitos foram definidos quatro componentes que serão responsáveis pelo funcionamento do *framework*. Esses componentes são o **Gerenciador de Eventos**, que é responsável por processar os eventos ocorridos durante a execução do framework; o **Gerenciador de Contextos** que monitora a condição atual do ambiente indicando qual é o contexto atual; o **Gerenciador de Comunicação** que é responsável por gerenciar a integração entre o framework e os provedores externos; o **Gerenciador de Dados** que é responsável por armazenar os dados da trilha.

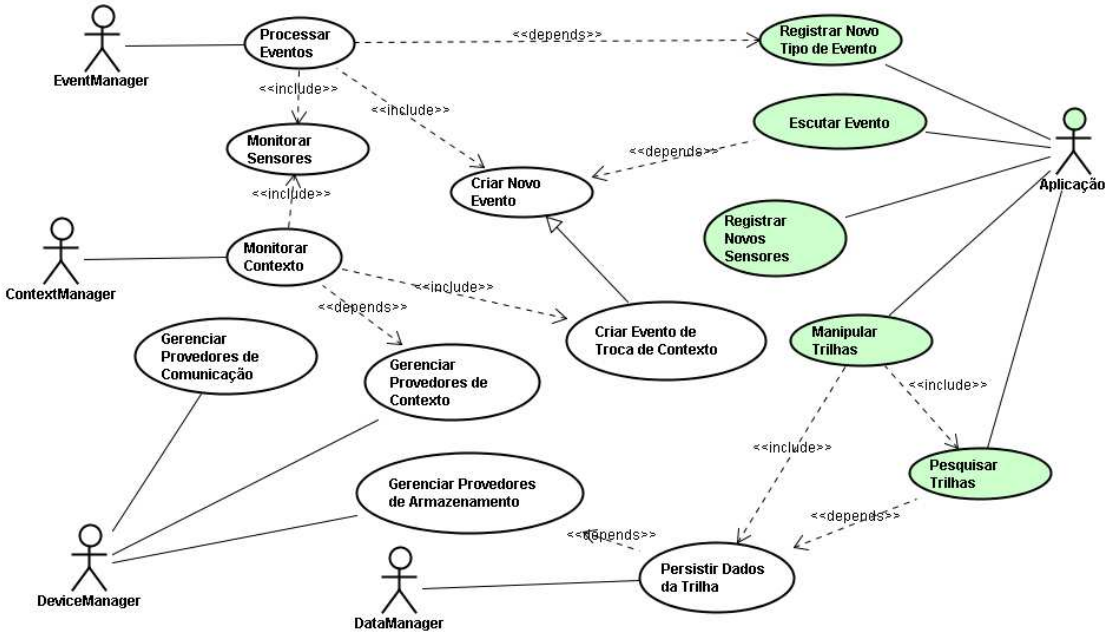

Figura 9. Requisitos funcionais do FrameTrail

Além disso, também estão relacionadas quais são as ações que as aplicações feitas com base no FrameTrail podem executar e como elas se relacionam com os demais requisitos. A seguir serão descritos os principais requisitos do *framework*:

1. **Monitorar Sensores:** Para possibilitar a sensibilidade ao contexto, e até mesmo a geração de eventos específicos de cada tipo de aplicação, o *framework* possui o suporte à parametrização de sensores que são encarregados de fazer a leitura de dados originados de *hardware* e *softwares* externos. Com base nisso, o *framework* se responsabiliza por monitorar esses sensores periodicamente, e mantém esses dados disponíveis para que os demais componentes possam acessá-los;

2. **Processar Eventos:** Esse é o requisito chave para o funcionamento do FrameTrail, pois o seu processo de funcionamento é baseado na criação de eventos. Com base nos tipos de eventos registrados, tanto em tempo de desenvolvimento quanto em tempo de execução, o *framework* se encarrega de verificar periodicamente quando as condições definidas por cada tipo de evento são satisfeitas e notifica todas as entidades envolvidas;

3. **Criar Novo Evento:** A criação de eventos ocorre quando as condições para a criação desse evento forem satisfeitas e alguma entidade estiver registrada para ser notificada por ele;

4. **Monitorar Contexto:** O processo de monitoramento do contexto é responsável por gerenciar as informações do ambiente atual e caracterizar a situação onde a entidade se encontra. A partir disso, as aplicações podem interagir com o ambiente e fazer uso dessas informações para auxiliar no controle dos dados das entidades envolvidas neles;

5. **Criar Evento de Troca de Contexto:** Cada vez que uma entidade entra em um contexto ou saí do contexto onde ela se encontra, um tipo de evento característico é criado. Processo de entrada em um contexto acontece quando todas as condições referentes aquele contexto forem satisfeitas. A saída de um contexto acontece quando qualquer uma dessas condições não for mais válida. Para tratar esses eventos, as aplicações podem monitorar toda a troca de contexto, ou apenas as entradas e saídas de algum contexto específico.

6. **Persistir Dados da Trilha:** É o processo de gravação de um novo *checkpoint* em uma trilha. Esse armazenamento pode ser feito tanto no dispositivo local onde a aplicação está executando, quando em um servidor remoto de dados. A seleção da estratégia desse processo é definida de acordo com a finalidade de cada aplicação. Em alguns casos a centralização dos dados é crucial, pois os dispositivos envolvidos não possuem capacidade para o armazenamento local, mas em outros casos a aplicação pode ser executada em um ambiente onde o uso de uma infraestrutura de servidores não se torna possível. O *framework* se propõe a dar suporte em ambos os casos, e até mesmo em situação híbridas, onde os dados da trilha são armazenados localmente e sincronizados com o servidor periodicamente.

Além dos requisitos envolvidos diretamente com os componentes do FrameTrail, existem outros que são desempenhados pelas aplicações desenvolvidas a partir dele. Esses requisitos estão destacados no diagrama e são os seguintes:

1. **Registrar Novo Tipo de Evento:** Cada aplicação trata de um domínio específico de dados e define o seu próprio processo de execução. Esse processo vai demandar etapas específicas para o seu funcionamento e essas etapas são tratadas como eventos pelo FrameTrail. Esses eventos podem ser baseados em dados captados pelos sensores do ambiente, ou por condições impostas pelo processo interno da aplicação. Nos dois casos a aplicação é capaz de registrar os tipos de evento que ela pretende trabalhar, e cada um desses tipos vai ser caracterizado por um conjunto de condições que devem ser satisfeitas para indicar sua ocorrência;

2. **Escutar Evento:** Uma aplicação pode se registrar para ficar esperando um ou mais tipos de eventos diferentes. Cada vez que um evento ocorre, todas as entidades que estão registradas para escutar ele são notificadas. Ao ser notificada à aplicação pode tomar alguma ação em função desse evento;

3. **Registrar Novos Sensores:** O processo de registro de novos tipos de evento pode demandar a análise de algum tipo de dado ainda não mapeada pelo

framework. Em função disso, cada aplicação pode registrar novos sensores que serão responsáveis por registrarem os dados lidos dos dispositivos físicos e notificar o framework das alterações desses dados;

4. **Manipular Trilhas:** Cada aplicação tem a possibilidade de determinar quando um novo *checkpoint* deve ser adicionado à trilha de uma entidade. Esse processo de criação de novo *checkpoint* pode ser feito de maneira explícita, ou seja, através da execução do serviço de salvamento de um novo *checkpoint* pela aplicação, ou de maneira implícita, que é feita pela ocorrência dos eventos definidos pela aplicação;

5. **Pesquisar Trilhas:** Além de criar *checkpoints* na trilha de uma entidade, as aplicações também necessitam pesquisar essas trilhas. Para isso o *framework* disponibiliza uma série de métodos de consulta que auxiliam as aplicações a selecionar um ou mais *checkpoints* do repositório de trilhas, independente de onde o repositório está (se é remoto ou local) e de como ele é implementado (seja através de bando de dados, arquivos XML ou objetos serializados).

Devido à diversidade dos ambientes onde as aplicações que trabalham com trilhas podem ser executadas, é importante que o *framework* siga os seguintes requisitos não funcionais:

1. **Flexibilidade:** Como as aplicações que suportam trilhas podem ser executadas em diversos tipos de dispositivos diferentes, tais como, computadores pessoais, servidores, *laptops*, *smartphones*, aparelhos de GPS entre outros, é imprescindível que o *framework* não possua restrições quanto às tecnologias utilizadas. Essa liberdade de implementações torna o *framework* independente de tecnologia. Outro aspecto importante é quanto à possibilidade de extensão de suas classes para criar aplicações de diversos domínios, pois isso proporciona o desenvolvimento de aplicações das mais variadas áreas de negócio;

2. **Sensibilidade ao contexto:** Dentro do modelo proposto, a trilha armazena as informações referentes aos contextos visitados por uma entidade. Sendo assim é imprescindível que o *framework* proporcione uma maneira para que as aplicações desenvolvidas a partir dele tenham o conhecimento do contexto onde a entidade está. Como a definição de contexto é muito ampla, pois caracteriza a situação atual da entidade como um todo e não apenas em relação ao posicionamento geográfico, é importante que as aplicações possam mapear novos tipos de informações que serão usadas durante a definição dos contextos;

3. **Suporte a Interatividade de serviços:** Devido à quantidade de dispositivos diferentes que podem ser usados nas aplicações que gerenciam trilhas, é comum que eles nem sempre tenham todos os recursos computacionais necessários. Para contornar as limitações inerentes aos dispositivos mais simples, é importante que seja possível delegar a execução das tarefas mais complexas para outros dispositivos presentes no ambiente de maneira dinâmica e transparente.

4.3 Arquitetura

Para seguir os requisitos especificados nesse trabalho, a arquitetura do *framework* foi projetada fazendo o uso de componentes que podem ser desenvolvidos em diversas plataformas diferentes. Para manter compatibilidade entre esses componentes, todo o núcleo do *framework* trabalha diretamente com uma camada de interface deixando a definição da implementação livre e sem restrições quanto à tecnologia e ambiente de execução.

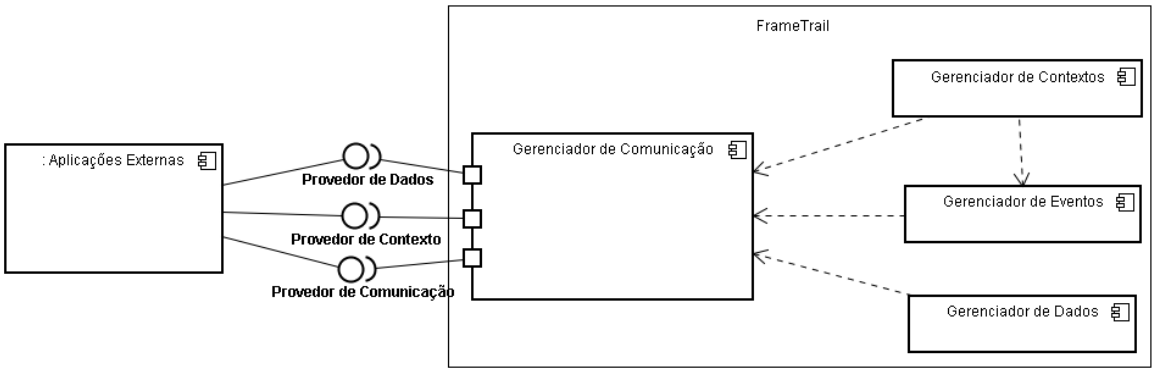

Figura 10. Arquitetura do FrameTrail

A Figura 10 mostra a organização desses componentes. As aplicações externas se integram com o *framework* através do uso de provedores específicos. Cada provedor é responsável por implementar uma funcionalidade em particular. Esses provedores são classificados em três grupos, os provedores de comunicação, provedores de contexto e os provedores de dados.

Os provedores de comunicação definem os canais de interação disponíveis para o funcionamento da aplicação e do *framework*. Esses canais servem para fazer a comunicação entre as entidades de maneira única e independente da plataforma em que elas estiverem sendo executadas. Através da definição dessa interface única, provedores desenvolvidos em diferentes tecnologias interagem entre si de maneira transparente. Por exemplo, uma aplicação usa um provedor de comunicação que funciona através de requisições HTTP, interage com outra aplicação que usa provedores baseados em *WebServices*. Neste caso o *framework* se encarrega de converter as mensagens do formato de entrada para uma representação própria em memória. Depois disso, ele processa a mensagem e envia os dados resultantes para a aplicação destino, no seu formato apropriado.

Os provedores de contexto são responsáveis por fazer a integração dos sensores externos com a infraestrutura do *framework*. Devido à grande diversidade de informações que podem ser usadas para definir um contexto, o *framework* deixa a cargo de cada aplicação definir quais provedores serão usados nas suas instâncias. Um provedor de contexto é responsável por monitorar uma informação específica, ou seja,

existem provedores de contexto que monitoram a data, a hora e a localização geográfica atual de uma entidade.

Os provedores de dados são responsáveis por gravar e recuperar as informações das trilhas salvas pelo *framework*. Essas operações podem ser executadas de diferentes maneiras, dependendo do objetivo de cada aplicação. Cada provedor implementa as operações de acesso a dados de uma maneira específica, existem provedores que armazenam as trilhas em arquivos XML, outros salvam em banco de dados externos.

A arquitetura baseada em provedores externos de funcionalidades possibilita que cada instância do *framework* defina sua maneira de trabalho sem afetar as aplicações desenvolvidas. Por exemplo, uma aplicação de monitoramento de corrida consiste em gravar o percurso percorrido por um grupo de pessoas e analisar o desempenho individual de cada participante. A aplicação monitora o deslocamento do usuário, salva os checkpoints e ao final mostra um resumo do seu percurso. Através da parametrização do *framework* é possível executar essa aplicação salvando os dados localmente, ou é possível também executar ela salvando os dados em um banco de dados central, alterando apenas o seu provedor de dados.

Além dos provedores externos, o *framework* é organizado internamente em quatro componentes específicos. Cada componente é responsável por gerenciar uma das etapas do processo de execução do *framework*. O Gerenciador de Eventos é encarregado de criar os eventos previamente parametrizados. Os eventos são parametrizados por tipos e cada tipo de evento define as triggers responsáveis pela criação de um novo evento. Essa parametrização pode ser feita durante a inicialização da instância do framework ou durante o processo de execução de uma aplicação.

O Gerenciador de Contextos é o componente responsável por detectar o contexto atual e tornar as suas informações acessíveis durante a execução das aplicações. O processo de reconhecimento de um contexto é feito através do monitoramento dos sensores cadastrados. Cada sensor mede algum tipo de informação e essas informações ajudam a definir um contexto através das restrições

definidas por ele. Quando o gerenciador de contexto identifica a troca de um contexto pela entidade que ele está monitorando, um evento específico é gerado notificando todas as entidades envolvidas.

O Gerenciador de Dados é responsável por manter os dados da trilha que são gerenciados pelas aplicações. Esse componente é responsável por gravar e consultar os dados das trilhas através do uso dos provedores de dados externos. O Gerenciador de Comunicação é responsável por realizar a integração entre todos os componentes do *framework* e as aplicações desenvolvidas a partir dele.

A separação desses componentes teve como principio básico o reuso e a possibilidade de implementação de um mesmo componente em plataformas distintas sem a necessidade de alterar os demais. Sendo assim, o mesmo framework pode ser executado tanto em plataformas móveis (telefones celulares, PDAs, aparelhos de GPS veiculares e etc), quanto em computadores pessoais e servidores. Os provedores externos indicam qual estratégia e qual a tecnologia que a aplicação em questão pretende usar enquanto, enquanto os gerenciadores tratam do fluxo de execução do *framework*.

4.4 Considerações sobre o Capítulo

Neste capítulo foi apresentado o framework desenvolvido neste trabalho, o FrameTrail. Foram detalhados os seus conceitos, características e requisitos. Em função desses requisitos a sua arquitetura foi elaborada através do uso de componentes extensíveis e reutilizáveis, para que a implementação de cada componente possa ser alterada de maneira independente sem modificar a estrutura de funcionamento do mesmo.

Outro aspecto importante sobre a arquitetura do Frametrail é a possibilidade de integração entre aplicações que fazem uso do *framework*, mas que foram desenvolvidas em diferentes plataformas. Nesse caso, plataforma não trata apenas de sistema operacional, mais sim do ambiente computacional como um todo, seja ele

móvel, servidor ou um computador pessoal. No próximo capítulo é descrita a maneira como o *framework* foi implementado.

5 PROTÓTIPO DO FRAMETRAIL

Neste capítulo estão descritos os aspectos referentes à implementação do protótipo do *framework* definido por esse trabalho.

Como um dos principais objetivos do FrameTrail é definir uma maneira de trabalho única para o desenvolvimento de aplicações com suporte a trilhas, independente de plataforma, tecnologia e domínio, a sua implementação deve contemplar a diversidade de arquiteturas de dispositivos existentes no mercado.

Sendo assim, o FrameTrail foi desenvolvido utilizando-se da tecnologia Java (Sun, 1997), por ser uma linguagem de programação orientada à objetos robusta e com o foco na portabilidade e independência de plataforma. Para o desenvolvimento foi utilizada a versão 1.6 da JDK do Java juntamente com a IDE Eclipse. Visando melhorar a aceitação do *framework* pelos desenvolvedores, as nomenclaturas de classes e métodos foram feitas no idioma inglês e seguem as convenções adotadas pela linguagem Java.

Nas próximas seções desse capítulo serão apresentados os principais conceitos envolvidos no fluxo de execução das funcionalidades providas pelo *framework*. Dentre esses conceitos será apresentado a abstração de dados empregada pelo FrameTrail, assim como as estratégias utilizadas para atender os requisitos especificados na modelagem do *framework*.

5.1 Abstração de Dados do FrameTrail

A Figura 11 apresenta a abstração de dados do FrameTrail. Essa abstração engloba as classes básicas do *framework*, ou seja, ela foca nas classes que representam as entidades de negócio utilizadas pelo FrameTrail. Dentre elas podemos destacar que

as classes *Entity*, *Container*, *Trail*, *CheckPoint*, *GeographicRegion* e *GeographicLocation*, foram apresentadas na Visão Geral do presente trabalho (capítulo 4.1).

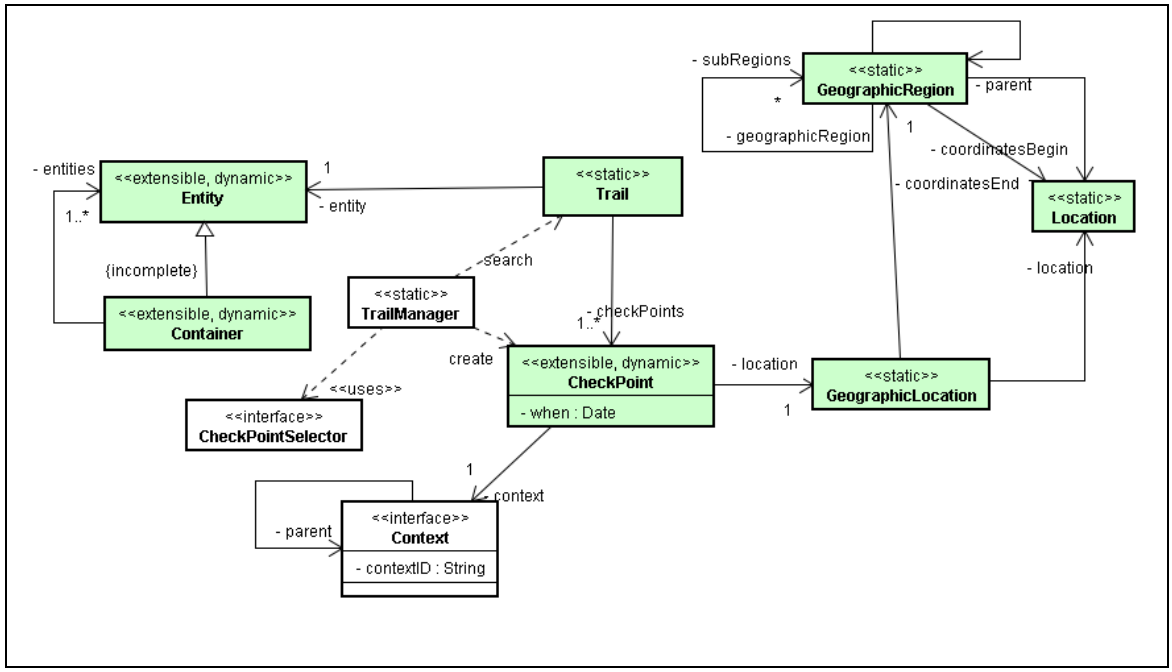

Figura 11. Abstração de dados do FrameTrail

Além delas foi criada também a classe *Location*, que representa as informações físicas de uma localização. Essas informações são geradas a partir do dispositivo de posicionamento utilizado pelo FrameTrail. Apesar de ser uma classe que define uma informação específica de um tipo de dispositivo, ela foi definida como uma classe básica porque ajuda a caracterizar fisicamente as regiões e localizações geográficas. Sendo assim, independentemente do tipo de dispositivo utilizado pelo *framework* para definir o posicionamento das entidades, as informações referentes à sua localização são tratadas de uma maneira única e integrada.

Outra classe importante dentro dessa perspectiva é a *TrailManager*, pois ela centraliza as funções relacionadas a criação de *CheckPoints*, assim como a vinculação desses *CheckPoints* a uma trilha (*Trail*). Além disso, essa classe implementa um mecanismo parametrizável de busca sobre os *CheckPoints* de uma trilha. Essa parametrização pode ser feita através da extensão da classe *CheckPointSelector*, que possui apenas um método que retorna um valor booleano indicando se o *CheckPoint* foi selecionado ou não.

5.2 Restrições

O conceito de Restrição (*Restriction*) no FrameTrail representa um critério a ser validado. Esses critérios são usados para definir as condições onde um evento ocorre ou quando um contexto está ativo. As restrições podem fazer uso dos dispositivos e sensores registrados no FrameTrail para a sua validação. A Figura 12 representa a estrutura das restrições definidas por padrão no FrameTrail.

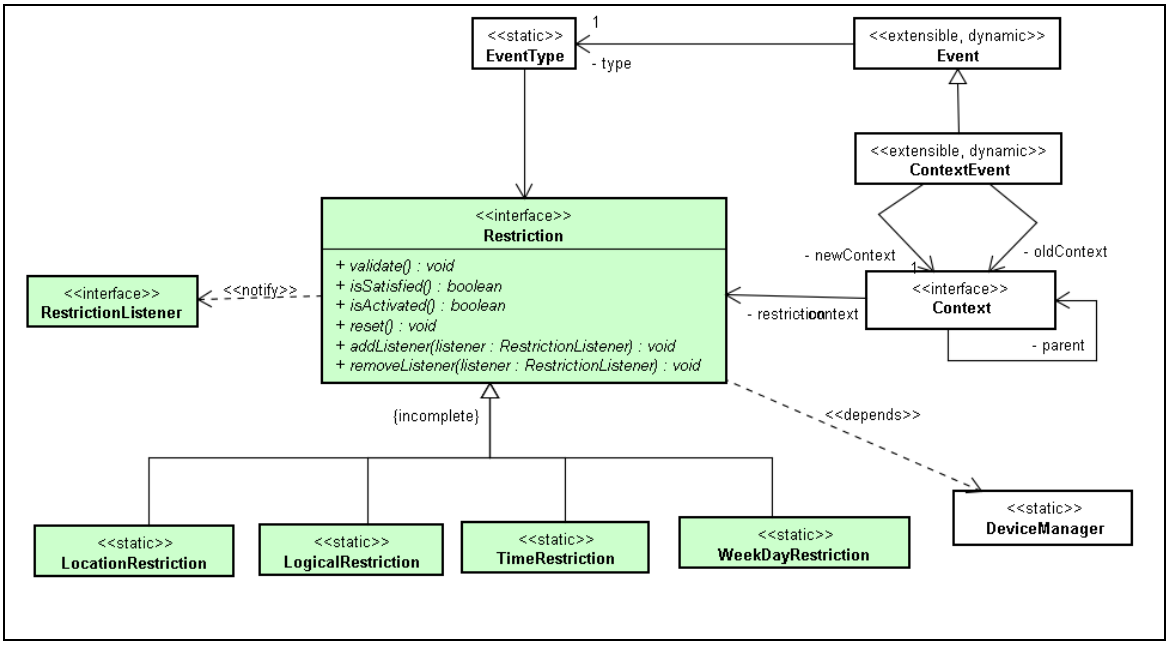

Figura 12. Restrições padrões do FrameTrail

Uma restrição pode estar vinculada a um ou mais *RestrictionListener.* Esses *listeners* são objetos que serão chamados quando uma restrição é ativada ou quando ela é desativada. Esses objetos são importantes, pois são eles que proporcionam a pró-atividade do *framework* para a criação de eventos (sejam eles de contexto ou não).

Outra característica importante das restrições é relativa à sua possibilidade de extensão. O FrameTrail traz um conjunto básico de restrições pré-definidas que servem de guia para a definição de novas restrições. Dentro dessa abordagem, se uma aplicação precisar de alguma lógica mais específica basta que seja criada uma nova classe que implemente a interface *Restriction* e que a mesma seja vinculada com o tipo de evento especifico (*EventType*) ou contexto (*Context*) que fará uso dela. O conjunto básico de restrições é definido pelas seguintes classes:

- ***LogicalRestriction*:** É composta por um operador e um conjunto de *Restrictions*. Essa restrição implementa os operadores OU lógico e E lógico. Sua validação consiste na execução do operador lógico contra todas as restrições vinculadas;

- ***LocationRestriction:*** Essa restrição faz uso do provedor de localização padrão definido durante a execução do *framework* e é ativado quando a entidade está dentro das coordenadas especificadas;

- ***TimeRestriction:*** Consiste na definição de dois horários limites (início e fim) e é ativada quando a hora atual do sistema encontra-se dentro desse limite;

- ***WeekDayRestriction***: Consiste na definição da lista de dias da semana, onde a restrição é ativada quando a data do sistema estiver contida dentro dessa lista.

5.3 Eventos e Contextos

O gerenciamento dos contextos, assim como o processamento dos eventos são funcionalidades chave para o funcionamento do FrameTrail. Enquanto o gerenciamento dos contextos é o requisito mínimo para a geração das trilhas, afinal a trilha representa os contextos visitados por uma entidade, a geração de eventos é a maneira que o FrameTrail faz para prover a comunicação pró-ativa entre as entidades.

Apesar de serem duas funcionalidades distintas, ambas se completam, pois o gerenciamento dos contextos é tratado como um tipo de evento específico (*ContextEvent*) e ele é gerenciado pelo mesmo mecanismo que controla os demais eventos do *framework*. A Figura 13 ilustra o relacionamento entre as classes que implementam essas funcionalidades.

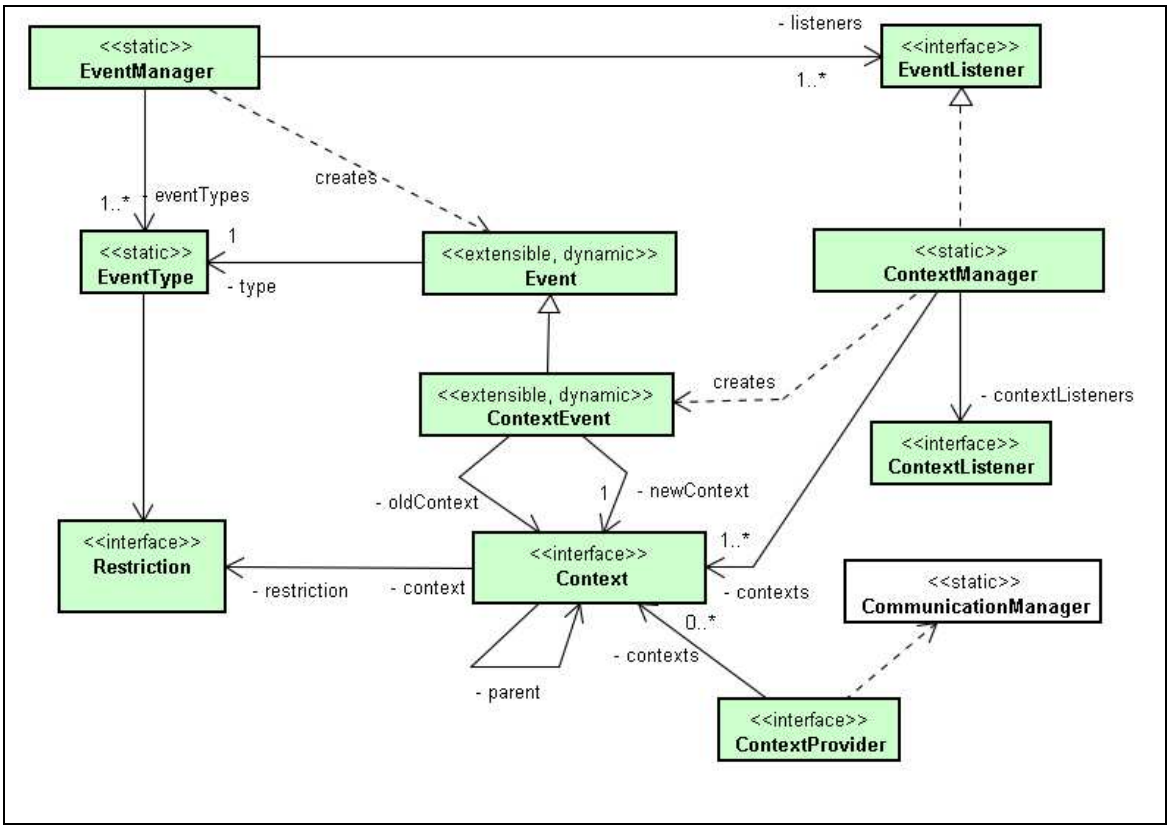

Figura 13. Eventos e Contextos do FrameTrail

A parametrização de eventos é feita através do registro de um novo *EventType* no *EventManager*. O *EventType* é a classe responsável por definir as características de um novo evento, ou seja, ele define as restrições que geram esse evento, assim como a classe especifica dele. Além disso, existe um método chamado *populateEvent*(*Event* evt), definido no *EventType*, que é chamado a cada vez que um novo evento é instanciado. Isso possibilita que o novo tipo de evento possa ser parametrizável com propriedades especificas dele, sem ter que alterar o mecanismo padrão de geração de eventos do FrameTrail.

Enquanto isso, o *EventManager* é responsável por fazer o monitoramento das restrições atribuídas a cada tipo de evento. Quando uma restrição é satisfeita, o *EventManager* se encarrega de criar o novo evento e notificar os *listeners* registrados.

A classe *ContextManager* implementa as funções referentes a definição dos contexto que serão monitorados pelo FrameTrail. Além disso, o *ContextManager* também gerencia e coordena o funcionamento dos *listeners* associados ao evento de

troca de contexto. Cada contexto registrado define um novo tipo de evento, que monitora o seu processo de entrada e saída das entidades. A classe *ContextProvider* define as características de cada contexto e seu funcionamento será detalhado na seção referentes aos provedores externos suportados pelo FrameTrail.

5.4 Provedores Externos

Outro aspecto importante de arquitetura do framework é a possibilidade de se implementar os provedores externos de maneira específica, dependendo de cada tipo de aplicação. Esses provedores são responsáveis por caracterizar o contexto no qual a aplicação está sendo executada, definem onde e como os dados serão salvos, além de proporcionar a comunicação entre diferentes tipos de aplicação. Esses provedores são separados em categorias, e cada uma delas será descrita nas seguintes sub-seções.

5.4.1 Provedores de Contexto

Os provedores externos de contexto são parte integrante do mecanismo de gerenciamento de contextos conforme descrito na seção 5.3. Como os contextos são definidos por características muito especificas e suas informações podem ter diferentes níveis de granularidade, dependendo do tipo de aplicação que está sendo desenvolvida, esses provedores possibilitam que cada aplicação defina os seus contextos conforme a sua necessidade.

Além de definir os dados básicos de cada contexto, tais como, identificadores, restrições, regiões geográficas e a sua hierarquia, também é possível definir provedores de dados e provedores de comunicação específicos para cada contexto. Isso possibilita a implementação de contextos remotos (quando as aplicações fazem uso de alguma estrutura computacional compartilhada), através do ajuste do provedor de comunicação durante a troca de contexto. Além disso, a definição de um provedor de dados diferente em um determinado contexto possibilita que as informações da trilha das entidades possam ser salvas em um banco de dados central, enquanto que

em outros contextos as informações serão salvas localmente, possibilitando assim integração total com aplicações desenvolvidas sob diferentes arquiteturas.

5.4.2 Provedores de Dados

Os provedores de dados são representados pela classe *DataProvider* e são responsáveis por manter o repositório de dados. Esse repositório armazena as informações de entidades e trilhas. A Figura 14 ilustra o relacionamento entre os provedores de dados e suas classes de implementação.

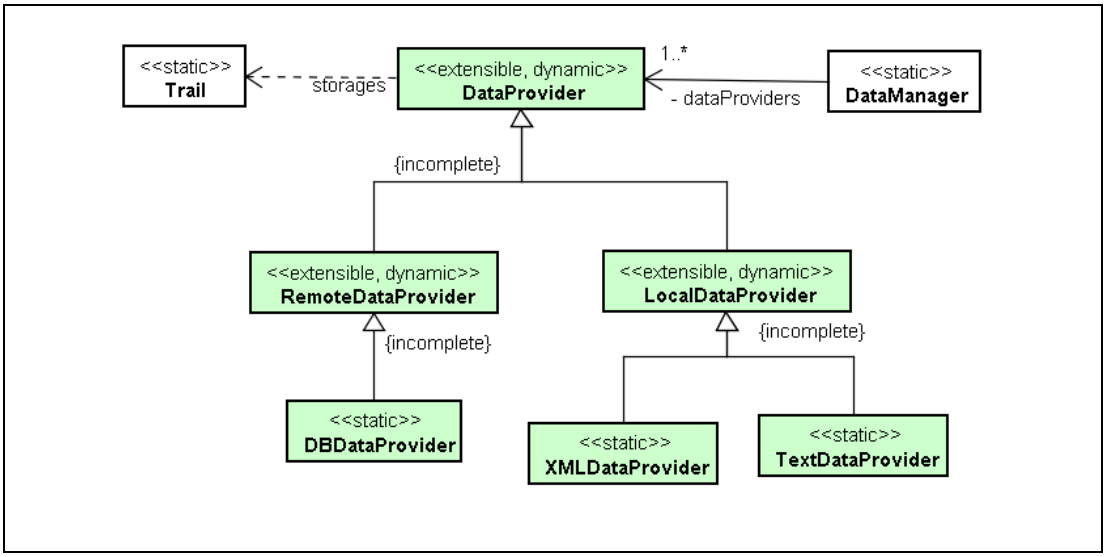

Figura 14. Provedores de Dados

A classe *DataManager* é responsável por controlar o ciclo de vida dos provedores cadastrados. O FrameTrail possui uma série de tipos provedores de dados previamente implementados que além de prover alternativas para o tratamento dos dados, também servem de exemplo para o desenvolvimento de futuras aplicações. Os provedores implementados são categorizados em dois grupos, e cada grupo representa uma estratégia diferente de onde os dados serão armazenados.

O primeiro grupo indica os provedores que armazenam os dados no computador onde o aplicativo está sendo executado. O segundo grupo engloba os provedores que armazenam os dados remotamente, ou seja, são aqueles que necessitam de algum tipo de conectividade de rede para executar suas operações. A classe *DataManager* se propõe a ser um catálogo com o registro de todos os

provedores disponíveis, mas a definição de quando executar cada provedor fica a cargo da aplicação.

5.4.3 Provedores de Comunicação

Os provedores de comunicação têm como principal objetivo integrar diferentes tipos de entidades independentemente da plataforma onde elas estejam sendo executadas.

Por exemplo, em um ambiente de execução onde exista uma infraestrutura computacional compartilhada, existe um servidor destinado a ser o canal de comunicação entre as entidades. Este servidor é capaz de fazer o intercâmbio de mensagens entre diferentes canais de comunicação, como por exemplo, uma aplicação envia uma mensagem através de uma conexão *Bluetooth* e o servidor a repassa para outra entidade através de uma conexão *http*. A Figura 15 mostra o diagrama com as classes que definem o mecanismo de registro e controle dos provedores de comunicação.

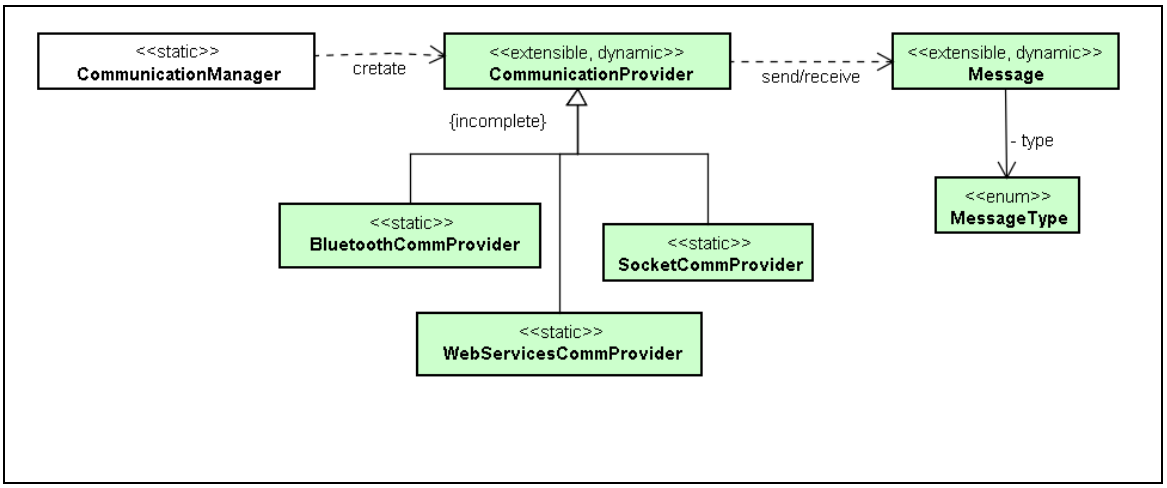

Figura 15. Provedores de Comunicação

Como definido no comportamento dos outros provedores externos, existe uma classe que centraliza o gerenciamento dos provedores cadastrados (*CommunicationManager*). Essa classe se encarrega de criar, iniciar e finalizar todos os provedores de comunicação cadastrados (*CommunicationProvider*).

A interoperabilidade entre diferentes tecnologias acontece através do uso da classe *Message*, que funciona como um descritor independente do tipo de conexão utilizado. Enquanto alguns provedores trafegam os dados como objetos, outros podem traduzir as mensagens como texto durante o seu processo de envio. Após o recebimento, o provedor converte novamente essa mensagem textual para um objeto da classe *Message* sem afetar o fluxo de funcionamento do FrameTrail.

5.5 Considerações sobre o capítulo

Esse capítulo apresentou os aspectos referentes à implementação do protótipo do FrameTrail. Dentre eles foram descritos a abstração de dados utilizada, ou seja, o relacionamento entre as classes do FrameTrail. Também foi descrito o funcionamento do mecanismo de restrições e como ele pode ser estendido. Além disso, também foi apresentado como funciona a definição de contextos e eventos. E por último foram apresentados os três tipos de provedores externos (Contextos, Dados e Comunicação) e o seu devido funcionamento.

O próximo capítulo apresenta a metodologia de avaliação utilizada para validar o FrameTrail.

6 AVALIAÇÃO

Tendo em vista a proposta apresentada pelo Frametrail, este capítulo apresenta a metodologia para sua avaliação e os resultados obtidos através desse processo.

6.1 Metodologia de Avaliação

Esse trabalho apresenta um *framework* que se propõe a facilitar e padronizar o desenvolvimento de aplicações com suporte a trilhas. A etapa de avaliação visa analisar o impacto do seu uso durante o processo de desenvolvimento dessas aplicações.

FrameTrail é um *framework*, portanto uma proposta de infraestrutura. Segundo (Edwards, 2003), a avaliação de uma infraestrutura é considerada problemática, uma vez que a mesma não é visível para o usuário. Os autores colocam em seu trabalho que somente é possível avaliar as funcionalidades de um *framework* construindo aplicações que as utilizem e então avaliá-las, assim, pode-se obter uma avaliação indireta do *framework*. Contudo, como observado por eles, ao construir uma aplicação deve-se ter em mente que os usuários não estarão julgando apenas as características que desejamos, mas o *software* como um todo.

Desta maneira, para a avaliação desta proposta, optou-se por realizar uma abordagem sob a perspectiva do desenvolvimento de *software.* Nesta abordagem, preza-se pela avaliação de características específicas do *framework* proposto e tem como objetivo avaliar o FrameTrail nos seguintes aspectos:

- **O *framework* suporta a construção de software com diferentes arquiteturas?** FrameTrail não impõe um modelo arquitetura para o desenvolvimento de

aplicações, pelo contrário, possibilita que o *framework* se adapte a cada tipo de aplicação. Assim, ele é maleável o suficiente para que o desenvolvedor crie sua própria organização;

Metodologia utilizada: desenvolvimento de um cenário de teste onde será construída uma aplicação que implementa os dois tipos de arquiteturas, ou seja, em determinados momentos à comunicação entre os dispositivos será feita através de um servidor centralizado (arquitetura cliente-servidor), e em outro momento a comunicação será feita diretamente entre as entidades (P2P);

- **O *framework* suporta a utilização de diferentes tipos de persistência para o armazenamento dos dados de trilhas?**

  Um dos requisitos do FrameTrail é a possibilidade de ser executado por diversos tipos de dispositivos e cada um desses tipos apresenta suas próprias limitações. Entre essas limitações está a capacidade de armazenamento de cada dispositivo. Sendo assim, o FrameTrail deixa a cargo de cada aplicação definir os provedores de dados mais adequados a sua realidade.

  Metodologia utilizada: construção de uma aplicação que armazene dados de trilha e que faça uso de diferentes provedores de dados para o armazenamento das trilhas. O framework deve possibilitar a troca desses provedores sem alterar a aplicação final.

- **O *framework* é customizável?**

  Este ponto tem como objetivo quantificar o esforço de programação necessário para estender o FrameTrail considerando novos casos de utilização. Esta métrica é dada pelos valores de:

  - quantas classes foram necessárias alterar;
  - quantas classes novas foram necessárias criar;

Metodologia utilizada: estes itens serão verificados durante a construção das aplicações de avaliação. Para tanto, serão obtidas informações do código-fonte final de cada uma delas.

A seguir serão descritos os estudos de caso utilizados para a execução da metodologia de avaliação apresentada.

6.2 Estudo de Caso 1: Educação Ubíqua

A aplicação dos conceitos da computação ubíqua em sistemas educacionais permite a construção de aplicações que são capazes de proporcionar a interação do aprendiz com o ambiente ao seu redor. Essa abordagem possibilita que as características desse ambiente possam ser vinculadas aos objetivos pedagógicos de cada aprendiz. Esse cenário deu origem a Educação Ubíqua (Barbosa, 2008), que tem o objetivo de permitir que o processo educacional ocorra em qualquer lugar a qualquer momento, independente do recurso utilizado. Segundo Silva (2009), o conceito de trilhas pode ser utilizado para auxiliar o processo de ensino e aprendizagem em um sistema de educação ubíqua. O registro das trilhas de um aprendiz por parte do sistema de educação ubíqua pode ajudar a determinar futuramente suas preferências e possíveis oportunidades pedagógicas.

Este estudo de caso consiste na simulação de uma aula virtual onde cada aluno faz uso de um dispositivo móvel para o acompanhamento dessa disciplina. Cada dispositivo executa o protótipo de um sistema educacional ubíquo, desenvolvido a partir das funcionalidades providas pelo Frametrail. Fazendo uso dessa aplicação, cada aluno poderá interagir com o professor e os demais colegas presentes em aula, para assim ter uma melhor experiência pedagógica. O objetivo deste cenário é demonstrar como as interações entre os alunos são representadas através do uso do conceito de trilha e como as abstrações definidas pelo FrameTrail podem ser empregadas durante esse processo.

6.2.1 Descrição do Cenário

O cenário utilizado para essa avaliação consiste em um curso presencial de programação usando a linguagem C. Esse curso acontece as segundas, quartas e sextas durante o período noturno e suas aulas são realizadas dentro do Laboratório de informática de uma Universidade. Para tornar a metodologia de ensino empregada nesse curso mais efetiva, a universidade disponibiliza para cada participante um computador portátil que será sua ferramenta de estudo durante o curso, auxiliando na execução e no gerenciamento das suas atividades.

Dentre as diversas funcionalidades que essa ferramenta educacional deve implementar, duas em específico serão abordadas no presente cenário. A primeira é referente à integração de dispositivos em diferentes ambientes, onde pode ou não haver uma infraestrutura computacional centralizada. Essa funcionalidade vai avaliar o *framework* quanto a sua capacidade de suportar o desenvolvimento de softwares feitos com diferentes tipos de arquiteturas.

No primeiro momento desse cenário, os alunos se encontram em sala de aula durante o horário do curso. Nesse ambiente a comunicação entre os dispositivos é realizada através de uma infraestrutura centralizada disponibilizada pela Universidade. Ou seja, as informações referentes ao contexto, assim como as trilhas geradas por cada entidade, são gerenciadas e armazenados em um servidor central.

Além dessa abordagem, também será avaliada a comunicação direta entre os dispositivos sem o uso de uma infraestrutura computacional centralizada. Para essa avaliação, o cenário a ser analisado consiste em um grupo de alunos realizando um trabalho fora do horário de aula. Esse grupo de estudos se reuniu em um sábado a tarde na casa de um dos integrantes para a realização dessa tarefa. Nesse caso as informações referentes ao contexto dos alunos são gerenciadas por cada dispositivo, assim como o armazenamento dos dados gerados por suas trilhas.

A segunda funcionalidade da aplicação avaliada dentro desse cenário é possibilidade da publicação de objetos de aprendizagem em um determinado

contexto. Essa funcionalidade é executada através da criação e parametrização de eventos específicos que serão processados pelos dispositivos do contexto. Sendo assim, a cada novo objeto de aprendizagem que for disponibilizado em uma aula, todos os alunos serão notificados através de um tipo evento específico sobre a publicação desse objeto de aprendizagem.

A modelagem do protótipo foi feita através do mapeamento dos processos executados pelas funcionalidades apresentadas. Esse mapeamento teve como objetivo adaptar os conceitos definidos nesses processos com a abstração de dados proposta pelo FrameTrail.

Sendo assim, os tipos de entidade mapeados foram Aluno, Professor e Objeto de Aprendizagem. Os contextos foram definidos como os encontros referentes às aulas, assim como os grupos de estudo realizados fora do horário de aula, além das regiões geográficas mais significativas, como por exemplo, o campus da universidade, o laboratório de informática e a biblioteca. Os tipos de eventos identificados foram os seguintes:

- **Criação de um novo grupo de estudo:** Este evento sinaliza para um conjunto de alunos que existe um novo contexto disponível. Esse contexto representa um grupo de estudo que pode ser disponibilizado em qualquer lugar e funciona de maneira descentralizada, provendo assim a comunicação direta entre os dispositivos de cada aluno;

- **Publicação de um novo Objeto de Aprendizagem:** Cada contexto, sendo ele uma aula ou um grupo de estudo, possui uma fila de objetos de aprendizagem disponível. Quando algum aluno ou professor adiciona um novo objeto nessa fila, todas as entidades do contexto são notificadas;

- **Início de uma aula:** No caso dessa aplicação, o início de uma aula é acionado automaticamente quando o aluno e encontra na sala onde a disciplina é lecionada, dentro do horário e nos dias da semana pré-estabelecidos. Nesse

caso o contexto é centralizado e as informações referentes às trilhas do usuário são salvos em um servidor da universidade;

- **Fim de uma aula:** Indica o término da aula. Nesse momento os dados da trilha do usuário são salvos no seu dispositivo pessoal, informando o resumo da aula em questão.

Segundo a abordagem desse estudo de caso, a trilha de um aluno vai indicar os momentos onde ele esteve envolvido em alguma atividade pedagógica, seja durante as disciplinas do curso, ou durante um grupo de estudos ou até mesmo ao acessar um objeto de aprendizagem. Sendo assim, a trilha de cada aluno será um conjunto de *checkpoints* agrupados em três tipos básicos. O primeiro é referente à publicação de um novo objeto de aprendizagem, onde além dos dados padrões de contexto, também será vinculado o objeto de aprendizagem que foi publicado. Os outros dois tipos de *checkpoints* são referentes às disciplinas cursadas e aos grupos de estudos que o aluno participou. O comportamento desses dois tipos de *checkpoints* segue o mesmo padrão, ou seja, será criado um *checkpoint* ao inicio e outro ao término de cada disciplina ou grupo de estudo, respectivamente.

Na próxima seção será apresentado o EduTrail, o protótipo desenvolvido para a validação desse cenário, e logo após serão discutidos os itens pré-definidos na metodologia de avaliação do FrameTrail.

6.2.2 Protótipo do EduTrail

A Figura 16 apresenta o diagrama de classes gerado a partir do desenvolvimento do protótipo do EduTrail. As classes destacadas no diagrama representam o conjunto de classes desenvolvido pela aplicação, enquanto que as outras são as classes padrões do FrameTrail e estão no diagrama apenas para contextualizar as classes do EduTrail.

A classe *EduTrailContextProvider* é a classe responsável por criar os dois tipos de contexto do EduTrail. É durante o processo de criação desses contextos que ocorre o vínculo deles com os seus respectivos provedores de dados ou comunicação. Por

exemplo, a classe *GrupoEstudoContext* é vinculado a um provedor de comunicação ponto a ponto, como por exemplo o *BluetoothCommProvider* (definido pelo FrameTrail mas omitido do diagrama com a finalidade de deixar o diagrama mais claro). Enquanto isso, a classe *DisciplinaContext* está vinculada com um provedor de comunicação centralizado, como por exemplo o *WebServicesCommProvider* (também omitido para melhor legibilidade do diagrama).

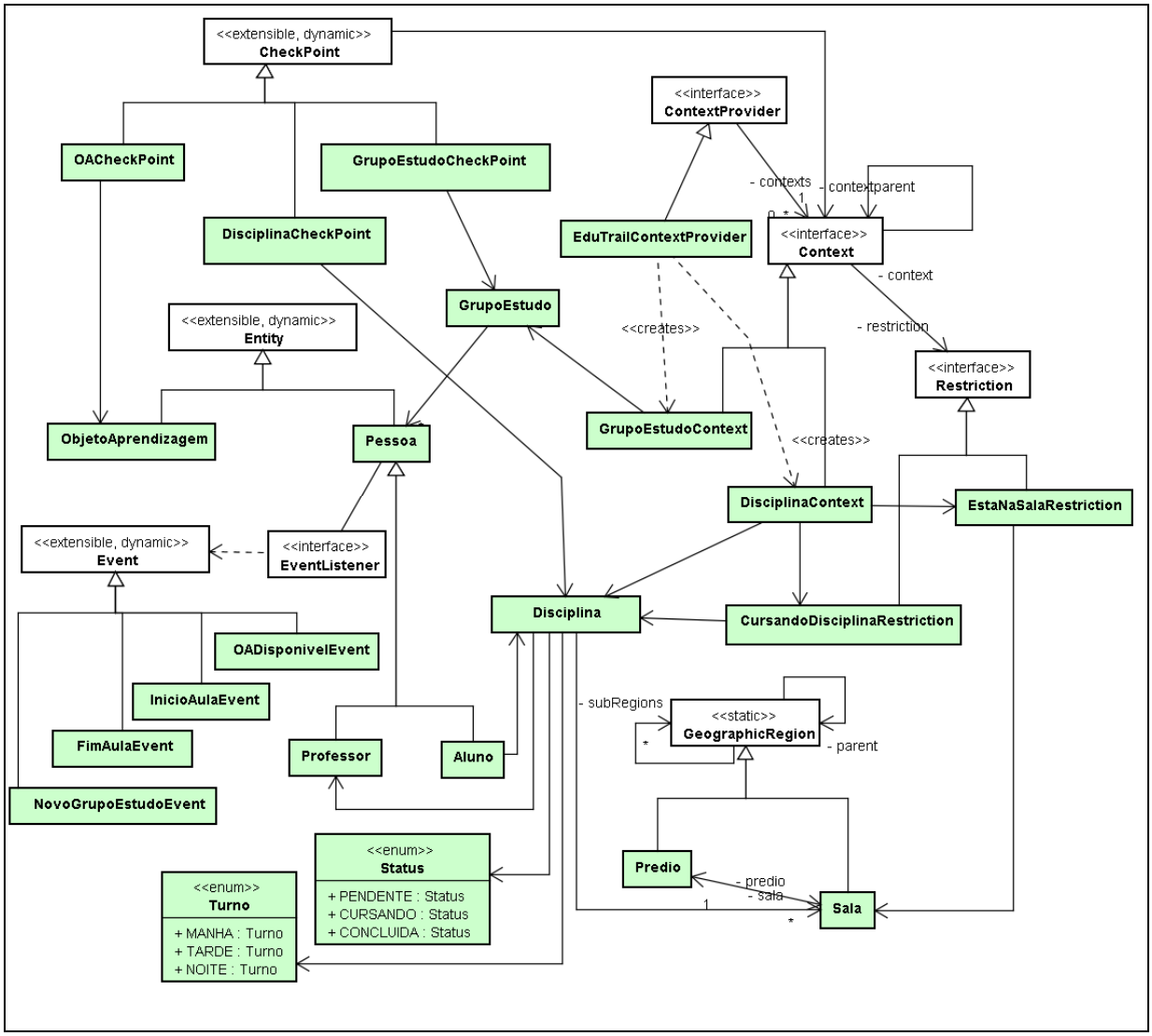

Figura 16. Diagrama de classes do EduTrail

Para a definição do contexto referente a uma disciplina, o EduTrail especializou duas restrições específicas para o seu funcionamento. A primeira é a *CursandoDisciplinaRestriction* que indica se um aluno esta devidamente matriculado em uma disciplina específica. A segunda restrição é a *EstaNaSalaRestricion*, que indica

se a entidade em questão (Aluno ou Professor) encontra-se dentro de uma sala pré-determinada da Universidade. Para fins de simulação desse aplicativo, os dados referentes ao sensor de localização são alimentados a partir de um arquivo texto com as coordenadas atuais da entidade.

Os prédios e as salas da universidade foram mapeados como classes filhas da classe *GeographicRegion* para que elas fossem devidamente vinculadas aos *Checkpoints* e contextos manipulados pela aplicação. As entidades Aluno e Professor foram especializações da classe Pessoa, que implementa o *listener* para os quatro tipos de eventos criados pelo EduTrail. O Objeto de Aprendizagem foi criado como uma classe filha a classe *Entity* para que fosse possível gravar a sua trilha. A trilha de um objeto de aprendizagem consiste nos momentos onde esse objeto foi acessado por alguma pessoa.

Para a definição da trilha de cada aluno, foram criados três novas classes que definem os *checkpoints* específicos de *EduTrail*, são elas: *OACheckPoint*, *DisciplinaCheckPoint* e o *GrupoEstudoCheckPoint*. Essas classes são subtipos da classe *CheckPoint*, e além de armazenar os dados padrões gerados pelo FrameTrail, elas também estão vinculadas aos objetos que as definem.

Por exemplo, a classe *OACheckPoint* é criada quando um aluno ou professor disponibiliza um novo objeto de aprendizagem para o contexto onde ele se encontra. Sendo assim, além dos dados que caracterizam o momento em que o *checkpoint* foi criado, também será vinculado a ele qual objeto de aprendizagem foi publicado naquele instante. As outras duas classes seguem o mesmo princípio e no momento de sua criação são vinculadas a sua disciplina ou ao seu grupo de estudo, respectivamente. Além disso, as classes *DisciplinaCheckPoint* e *GrupoEstudoCheckPoint* também armazenam a informação do momento em que elas foram criadas, ou seja, indica se o *checkpoint* representa o inicio ou o término da disciplina ou grupo de estudo, respectivamente.

Como resultado do software especificado acima, foi gerado um protótipo da ferramenta EduTrail, conforme ilustra a Figura 17.

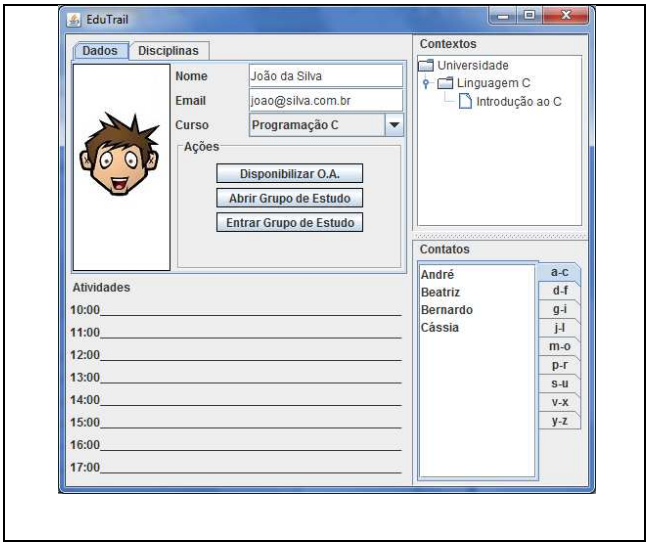

Figura 17. Protótipo do EduTrail

A tela do EduTrail é separada em quatro seções: a) o painel central superior com os dados do aluno, as disciplinas matriculadas e as ações que ele pode executar; b) o painel central inferior que lista as atividades do aluno; c) o painel lateral superior que apresenta a árvore de contextos ativos; d) o painel lateral inferior que traz a lista de contatos do aluno.

Toda vez que o usuário se desloca entre os contextos, a árvore de contextos ativos é atualizada com a hierarquia do contexto atual, e a lista de contatos também é atualizada para exibir apenas quem está acessível naquele contexto. Nesse momento um novo *checkpoint* é criado indicado e entrada no contexto ativo. Caso esse contexto seja uma disciplina ou um grupo de estudos, serão criados os *checkpoints* específicos, de acordo com o comportamento detalhado anteriormente nesse capítulo.

O botão de ação “Disponibilizar O.A.” permite que o usuário compartilhe um dos seus objetos de aprendizagem com todas as pessoas que estão vinculadas ao seu contexto atual. Quando um objeto de aprendizagem é disponibilizado, o sistema gera um novo *checkpoint* indicando esse evento. Enquanto isso, o botão “Abrir Grupo de Estudo” cria um novo contexto onde o usuário pode selecionar os contatos que ele quer convidar para fazer parte dele. O botão “Entrar Grupo de Estudo” possibilita que o aluno ingresse em um dos grupos de estudo que ele foi convidado. A Figura 18 ilustra um exemplo de uma trilha de um aluno, gerada apartir das definições do EduTrail.

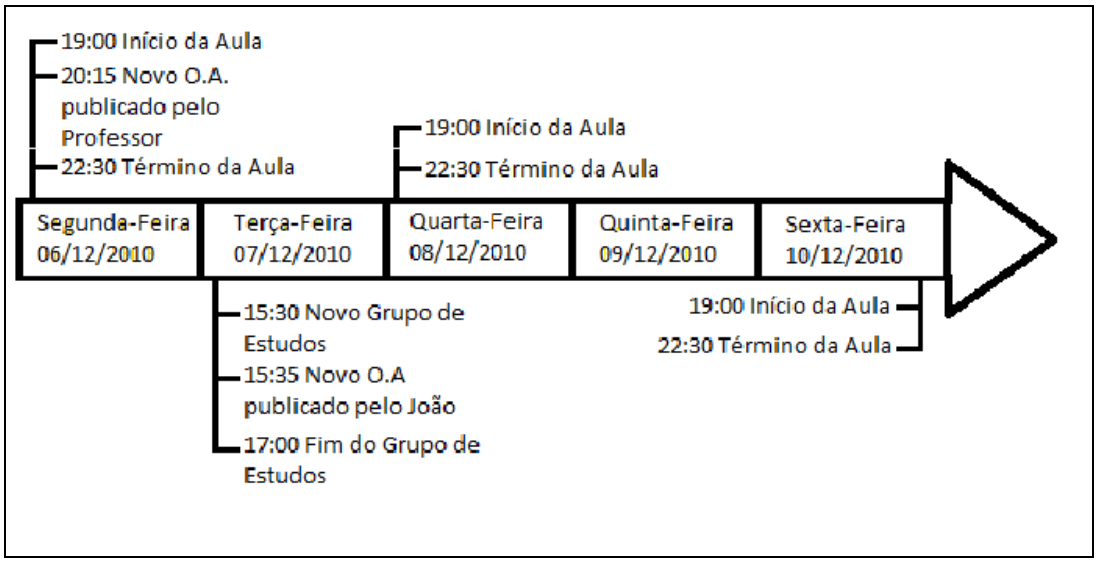

Figura 18. Exemplo de Trilha do EduTrail

Para agilizar o processo de testes do EduTrail, foi criado uma classe auxiliar que se encarrega de montar todo o ambiente de testes a partir de um arquivo texto. Nesse ambiente de testes podem-se definir os alunos, professores, objetos de aprendizagem, disciplinas, contexto, salas e prédios utilizados durante execução da aplicação.

6.2.3 Avaliação do FrameTrail

Esta seção apresenta os resultados obtidos a partir da avaliação do FrameTrail durante o processo de desenvolvimento do EduTrail. A seguir serão descritos os quesitos avaliados e como o FrameTrail foi usado para atendê-los.

6.2.3.1 O framework suporta a construção de software com diferentes arquiteturas?

Considerando o funcionamento do EduTrail, pode-se afirmar que o FrameTrail suporta o desenvolvimento de software em diferentes arquiteturas. Conforme descrito anteriormente, o EduTrail se caracteriza por gerenciar atividades educacionais em dois momentos distintos.

O primeiro momento acontece durante os encontros presenciais das disciplinas. Nesse instante o EduTrail trabalha através da arquitetura cliente-servidor, pois as entidades vinculadas ao contexto atual são gerenciadas pela infraestrutura provida pela universidade. Sendo assim, um servidor central gerencia a comunicação

entre as entidades que estão nesse contexto, além de armazenar os dados referentes as trilhas dessas entidades.

No segundo momento, os alunos encontram-se fora da universidade, e fazem uso do EduTrail sem uma infraestrutura compartilhada. Nesse caso, a aplicação apresenta características de um arquitetura ponto a ponto, onde a comunicação é feita diretamente entre as entidades. Além disso, as informações referente as trilhas das entidades são armazenadas localmente.

A classe que implementa esse comportamento é a *EduTrailContextProvider*, pois ao criar os contextos das disciplinas e dos grupos de estudo, ela se responsabiliza em registrar os provedores específicos para cada tipo de contexto. Com isso, a aplicação não precisa implementar controles para o gerenciamento desse tipo de ação, pois a comunicação entre entidades feita através dos eventos gerados, assim como as operações de manipulação dos dados das trilhas são executadas em função do contexto atual.

6.2.3.2 O framework suporta a utilização de diferentes tipos de persistência para o armazenamento dos dados de trilhas?

Considerando o funcionamento do EduTrail, pode-se afirmar que o FrameTrail suporta a utilização de diferentes tipos de persistência para o armazenamento de dados de trilhas. A realização desse quesito acontece da mesma forma do demonstrado no quesito anterior. No momento que o *EduTrailContextProvider* cria os contextos, ele vincula diferentes tipos de provedores de dados para cada um deles.

Através desse comportamento, o *EduTrail* faz uso de dois provedores de dados diferentes. O primeiro é o *DBDataProvider*, que é utilizado nos contextos das disciplinas e os dados referentes as trilhas são armazenados em um servidor remoto. O segundo é utilizado quando os alunos não se encontram na universidade, ou seja, não estão em um contexto que possui uma infraestrutura compartilhada. Nesse caso os dados são manipulados através do *XMLDataProvider*, que armazena os dados localmente.

A classe *DataManager* possibilita o registro de provedores de dados. Um contexto pode ou não estar vinculado a um provedor de dados. Sendo assim, a aplicação não precisa se preocupar com o processo de seleção do provedor de dados a ser usado, pois toda vez que ela for manipular os dados das trilhas, o *DataManager* vai fornecer a instancia correta do provedor de dados em função do contexto ativo.

6.2.3.3 O framework é customizável?

Considerando o processo de desenvolvimento do EduTrail, pode-se identificar que o FrameTrail é um framework passível de customização. O diagrama de classes representado pela Figura 16 foi analisado para a avaliação desse quesito. Esse diagrama é representado por trinta classes diferentes. Dessas, oito classes (*CheckPoint*, *ContextProvider*, *Context*, *Entity*, *Event*, *EventListener*, *Restriction* e *GeographicGeographiRegion*) são classes do FrameTrail e estão no diagrama apenas para contextualização.

As vinte duas classes restantes podem ser categorizadas em 2 grupos distintos: as classes de domínio e as classes de integração. As classes de domínio representam objetos específicos para o domínio do EduTrail, ou seja, são as classes envolvidas durante o processo de gerenciamento das atividades educacionais. Esse grupo é composto por 10 classes (*Pessoa*, *Professor*, *Aluno*, *ObjetoAprendizagem*, *GrupoEstudo*, *Disciplina*, *Status*, *Turno*, *Sala* e *Predio*).

O segundo grupo é composto pelas classes de integração, ou seja, são as classes responsáveis por adaptar as funcionalidades FrameTrail ao domínio de aplicação do EduTrail. Esse grupo é composto por doze classes, que podem ser divididos pelos conceitos do FrameTrail, portanto elas são compostas por três checkpoints (*OACheckPoint*, *DisciplinaCheckPoint* e *GrupoEstudoCheckPoint*), dois contextos (*DisciplinaContext* e *GrupoEstudoContext*), duas restrições (*CursandoDisciplinaRestriction* e *EstaNaSalaRestriction*), quatro eventos (*OADisponivelEvent*, *InicioAulaEvent*, *FimAulaEvent* e *GrupoEstudoEvent*) e um provedor de contexto (*EduTrailContexProvider*).

O FrameTrail pode ser considerado um *framework* passível de customização, pois não foi necessário alterar nenhuma de suas classes internas para o desenvolvimento do EduTrail. A integração entre a aplicação e o *framework* foi feita através da especialização dos conceitos do FrameTrail pelo EduTrail.

6.3 Estudo de Caso 2: Transporte de Cargas

O transporte de cargas é uma atividade rentável, lucrativa e extremamente competitiva. É graças a esse serviço que os produtos produzidos pelas indústrias chegam ao comércio varejista, que por sua vez tem o papel de levá-los até o consumidor final. A cada dia que passa o consumidor se torna cada vez mais exigente quanto à qualidade do produto oferecido. Além de uma qualidade acima da média, o consumidor também procura o preço mais acessível dentre os fornecedores de um determinado tipo de produto.

Dentro desse cenário, é cada vez mais comum ver grandes empresas investirem em sistemas de logística cada vez mais avançados, visando otimizar os seus prazos de entrega, assim como melhorar o controle da qualidade do produto a ser transportado. Toda a economia gerada através da adoção de processos mais efetivos, assim como controles mais precisos, podem ser passados ao preço final do produto fazendo com que ele tenha uma vantagem competitiva sobre os demais concorrentes.

Esse estudo de caso consiste na simulação de um sistema de logística sensível ao contexto e com suporte a trilhas. Dentro desse cenário a entidade central é a carga a ser transportada. Essa carga pode ser transportada por navio ou por caminhão, em alguns casos, pelos dois tipos de transporte. O objetivo desse cenário é demonstrar como os conceitos do FrameTrail podem ser empregados no monitoramento do deslocamento dessa carga, assim como as condições em que essa carga está sendo transportada.

6.3.1 Descrição do Cenário

O cenário utilizado para essa avaliação consiste no sistema de logística de um frigorífico situado no sul do Brasil e que exporta carne congelada para os Estados Unidos e para a Europa. Devido às grandes distâncias percorridas para a entrega da carne, a viagem é divida em etapas onde cada etapa representa um tipo de transporte.

Neste cenário a viagem pode ser feita tanto por modal marítimo quando por modal rodoviário. A troca de modal só acontece em um Porto, onde a carga passa do caminhão para o navio ou vice-versa. Dentro de uma mesma viagem o contêiner que transporta a carga pode ser descarregado em mais de um local, cada local de descarregamento é chamado de Centro de Distribuição. É nos centros de distribuição que a carga bruta vinda do fornecedor é divida e distribuída entre os comerciantes de menor porte.

Para manter o rastreamento total da carga, cada carregamento de carne é feito dentro de um mesmo contêiner durante toda a sua viagem. Esse contêiner possui sensores referentes à temperatura e peso da carga, assim como a inclinação do contêiner. A utilização desses sensores visa o monitoramento da qualidade da carne a ser entregue, pois como a carne é transportada de forma congelada um aumento significativo de temperatura pode danificar o produto. O sensor referente ao peso é usado para alertar em que momentos o contêiner foi descarregado, enquanto o sensor de inclinação monitora a maneira como a carga é içada do caminhão para navio ou vice-versa.

A funcionalidade provida pelo FrameTrail que será avaliada por essa aplicação, é o conceito de *Container*. Esse conceito tem como objetivo agregar entidades hierarquicamente a fim de compartilhar seu registro de trilhas. Nessa perspectiva, a carga é representada como uma entidade, pois é ela que terá as suas informações de trilha armazenadas. Já os meios de transporte (navio e caminhão) são implementados por classes do tipo *Container*, pois eles apenas compartilharam suas informações de contexto para complementar a trilha da carga.

Outra funcionalidade explorada por essa aplicação é o suporte do FrameTrail a contextos dinâmicos, pois cada viagem é representa por um contexto onde serão definidos uma localização de origem e outra de destino. Para o monitoramento mais efetivo do descolamento dos veículos, cada viagem é dividida em vinte sub-contextos chamados cercas eletrônicas. Cada cerca eletrônica representa uma área equivalente a cinco por cento da viagem total. Essa subdivisão de contextos proporciona um evento novo de alteração de contexto a cada etapa da viagem. Assim que uma das cercas eletrônicas é ultrapassada, o sistema cria automaticamente a cerca eletrônica referente à próxima etapa da viagem.

Na próxima seção será apresentado o CargoTrail, o protótipo desenvolvido para a validação desse cenário, e logo após serão discutidos os itens pré-definidos na metodologia de avaliação do FrameTrail.

6.3.2 Descrição do CargoTrail

A Figura 19 apresenta o diagrama de classes gerado a partir do desenvolvimento do protótipo do CargoTrail. As classes destacadas no diagrama representam o conjunto de classes desenvolvido pela aplicação, enquanto que as outras são as classes padrões do FrameTrail e estão no diagrama apenas para contextualizar as classes do CargoTrail.

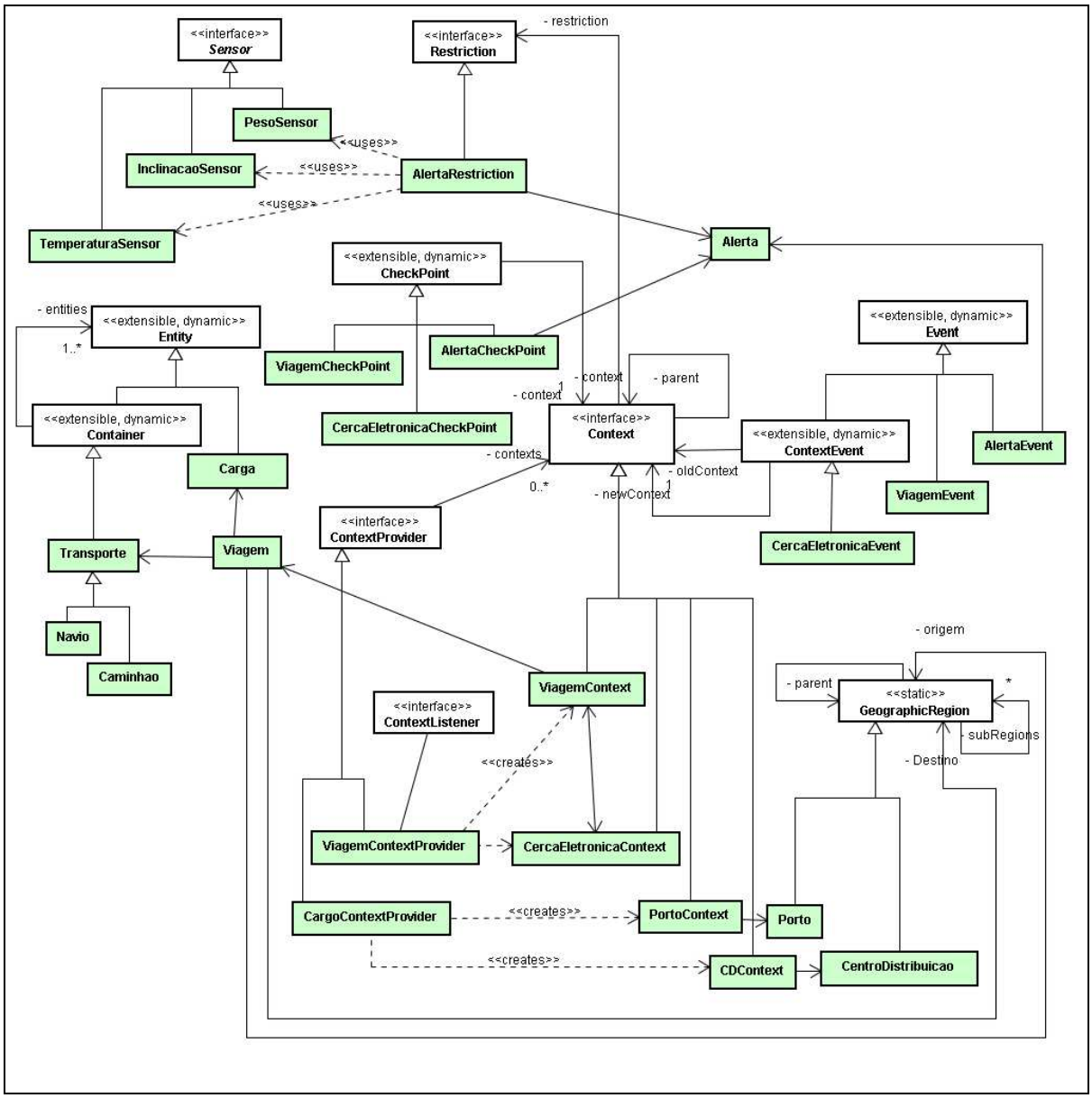

Figura 19. Diagrama de classes do CargoTrail

O CargoTrail trabalha com dois provedores de contexto. O primeiro provedor é o *CargoContextProvider* que é responsável por criar os contextos referentes aos portos (*PortoContext*) e aos centros de distribuição (*CDContext*). Esses contextos armazenam os dados gerados durante a viagem em um servidor central disponível em cada um deles. O segundo provedor é o *ViagemContextProvider*, que tem a função de criar os contextos referentes as viagens (*ViagemContext*) e as cercas eletrônicas (*CercaEleltronicaContext*). Esses dois tipos de contexto usam uma estratégia de armazenamento local das informações, mas a cada final de viagem, ou seja, quando carga chega a um porto ou centro de distribuição, os dados da trilha da carga são

armazenados no servidor e apagados do sistema de armazenamento local. Ao final de cada etapa, a empresa possui um relatório detalhado do andamento da viagem.

As entidades mapeadas para o CargoTrail foram a *Carga,* que estende a classe *Entity* e representa a entidade a ser trilhada pelo sistema, enquanto que as classes *Navio* e *Caminhão* estendem a classe *Transporte*, que por sua vez é filha da classe *Container*. Sendo assim, os dados da trilha referente ao transporte da carga serão adicionados as informações da sua trilha, gerando um retrato fiel do que aconteceu durante a viagem.

Além do deslocamento, o CargoTrail também se propõe a monitorar a qualidade dessa carga, sendo assim, ele trabalha com um sistema de alertas que monitoram os sensores definidos pela aplicação. Os sensores específicos utilizados pelo CargoTrail são o *PesoSensor*, que retorna o peso da carga em quilogramas, o *InclinacaoSensor*, que retorna a inclinação em graus radianos do contêiner onde a carga está e por ultimo está o *TemperaturaSensor*, que retorna a temperatura atual da carga em graus Celsius. Cada alerta é criado a partir de um valor base e possui uma taxa de tolerância, quando essa taxa de tolerância é excedida, a restrição *AlertaRestricion* é satisfeita e um novo evento do tipo *AlertaEvent* é lançado e o registro de um novo *checkpoint* (*AlertaCheckPoint*) é adicionado a trilha.

Os outros eventos gerenciados pelo CargoTrail são o *ViagemEvent*, que é lançado ao inicio ou término de uma viagem, enquanto que o *CercaEletronicaEvent* é lançado a cada troca de contexto, quando o novo contexto é derivado do *CercaEletronicaContext*.

Como resultado do software especificado acima, foi gerado um protótipo da ferramenta CargoTrail, conforme ilustra a Figura 20.

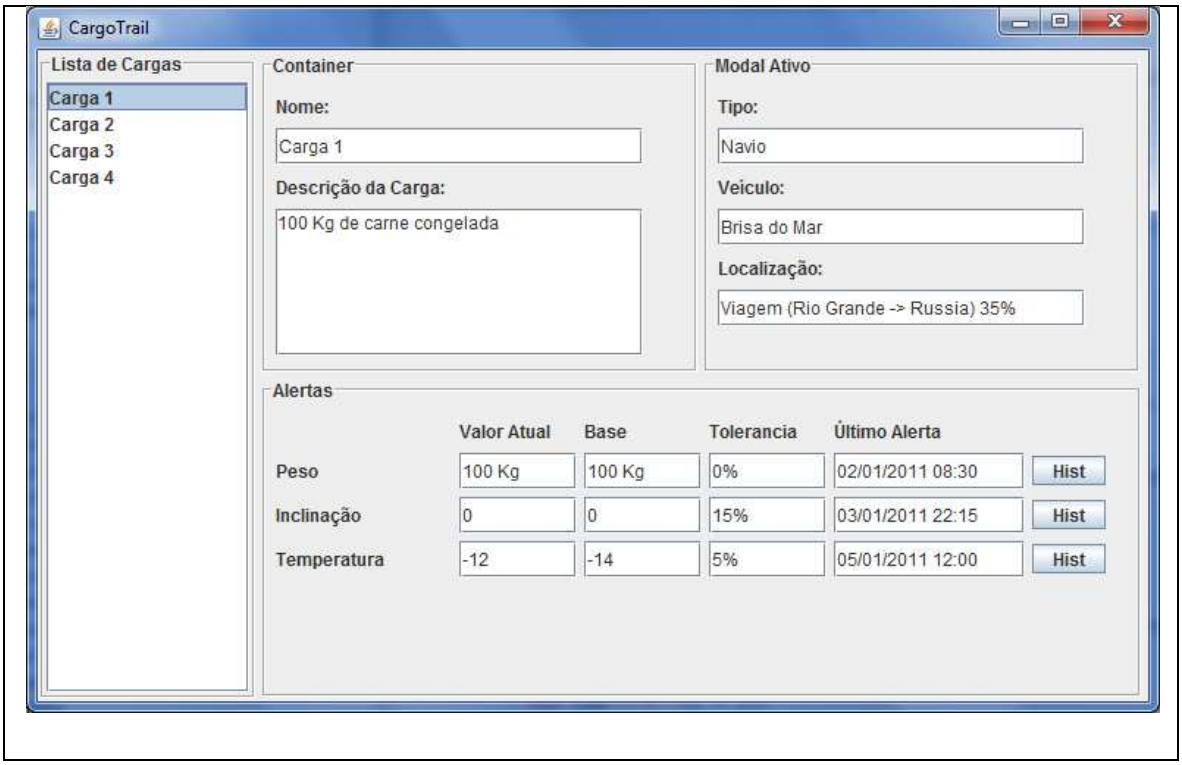

Figura 20. Protótipo do CargoTrail

A tela do CargoTrail é o painel geral das cargas que estão sendo monitoradas. Esse painel é dividido em quatro seções: a) a Lista de Cargas lateral, que indica todas as cargas que estão em deslocamento; b) os dados do Contêiner, indicando seu identificador e um breve resumo da carga transportada; c) o painel de modal ativo, que indica qual o meio de transporte que está levando a carga, assim como o veículo utilizado e o andamento total da viagem; d) painel de alertas que serve para monitorar os sensores acoplados a carga.

Para cada sensor é possível ver o seu valor atual (nesse caso o ultimo valor lido, pois essa tela é gerencial e já mostra os dados consolidados), valor base, ou seja, o valor que o esperado por esse sensor, a taxa de tolerância e a data e hora da última vez em que o alerta desse sensor foi emitido.

Sendo assim, a trilha de cada carga dentro de CargoTrail é representada pela a união dos seus dados de deslocamento, juntamente com os dados dos sensores que medem a qualidade da carga transportada. A Figura 21 ilustra como a trilha de uma carga é representada pelo CargoTrail.

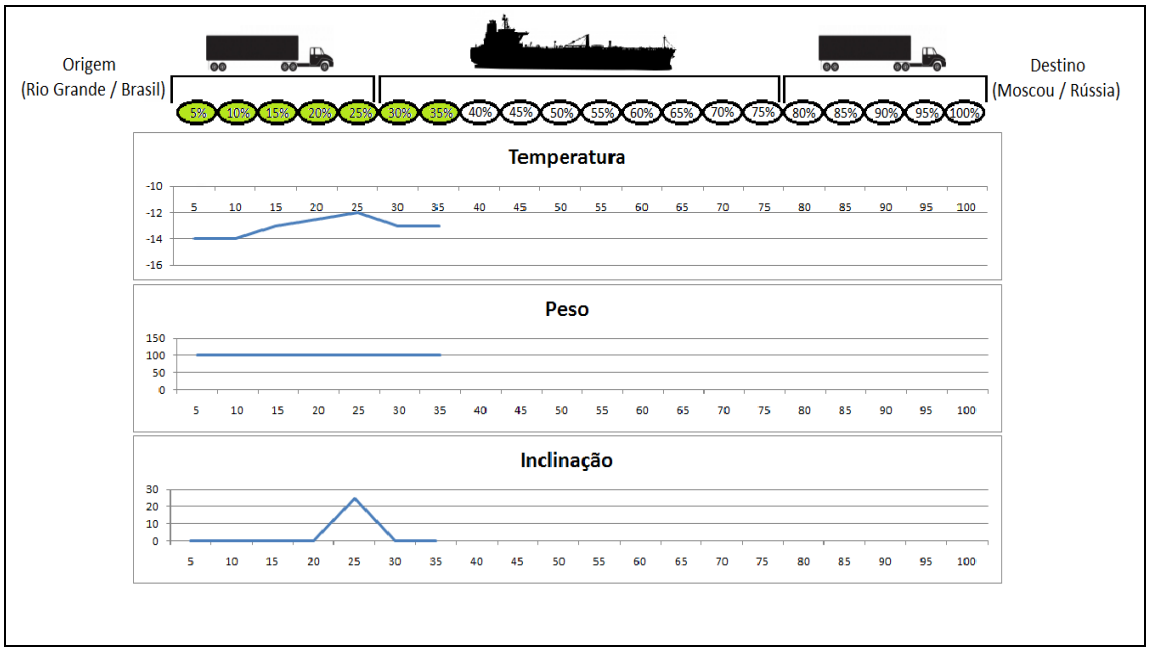

Figura 21. Exemplo de Trilha do CargoTrail

Para o processo de testes desse protótipo, tanto os sensores específicos do CargoTrail, como o sensor de localização provido pelo FrameTrail foram baseados na leitura de arquivos texto, visando assim agilizar o processo de avaliação do mesmo. Além disso, todos os dados referentes à execução desses testes foram carregados para o ambiente através de classes auxiliares que possibilitam a definição de Portos, Centros de Distribuição, Viagens, Cargas e meios de transporte (Navio e Caminhão), visando criar uma massa de testes mais ampla e assim melhorar o grau de significância dos testes.

6.3.3 Avaliação do FrameTrail

Esta seção apresenta os resultados obtidos a partir da avaliação do FrameTrail durante o processo de desenvolvimento do CargoTrail. A seguir serão descritos os quesitos avaliados e como o FrameTrail foi usado para atendê-los.

6.3.3.1 O framework suporta a construção de software com diferentes arquiteturas?

Considerando o funcionamento do CargoTrail, pode-se afirmar que o FrameTrail suporta o desenvolvimento de software em diferentes arquiteturas. Apesar do conceito de arquitetura P2P não ser aplicado para esse estudo de caso, vale ressaltar que durante a viagem as informações referentes a trilha da carga são

armazenadas localmente, ou seja, a aplicação não faz uso de uma infraestrutura compartilhada de recursos, fato que caracteriza uma arquitetura cliente-servidor.

Sendo assim, pode-se dizer que o CargoTrail roda tanto com a arquitetura cliente-servidor (no caso de estar em um Porto ou Centro de Distribuição) quando em uma arquitetura local, onde o sistema não se comunica com nenhuma outra aplicação, mas mantém o seu funcionamento normalmente.

6.3.3.2 O framework suporta a utilização de diferentes tipos de persistência para o armazenamento dos dados de trilhas?

Considerando o funcionamento do CargoTrail, pode-se afirmar que o FrameTrail suporta a utilização de diferentes tipos de persistência para o armazenamento dos dados de trilha. Nesse quesito a estratégia usada no CargoTrail foi a mesma utilizada no EduTrail (seção 6.2.3.2). A aplicação define quatro tipos de contexto diferentes, sendo que dois deles (*ViagemContext* e o *CercaEletronicaContext*) utilizam um provedor de dados local enquanto os outros dois (*PortoContext* e o *CDContext*) utilizam um provedor de dados remoto.

6.3.3.3 O framework é customizável?

Considerando o processo de desenvolvimento do CargoTrail, pode-se afirmar que o FrameTrail é um framework passível de customização. O diagrama de classes representado pela Figura 19 foi analisado para a avaliação desse quesito. Esse diagrama é representado por trinta e cinco classes diferentes. Dessas, onze classes (*Sensor*, *CheckPoint*, *ContextProvider*, *Context*, *ContextEvent*, *Entity*, *Container*, *Event*, ContextListener, *Restriction* e *GeographicGeographiRegion*) são classes do FrameTrail e estão no diagrama apenas para contextualização.

As vinte quatro classes restantes podem ser categorizadas em 2 grupos distintos: as classes de domínio e as classes de integração. As classes de domínio representam objetos específicos para o domínio do CargoTrail, ou seja, são as classes envolvidas durante o processo de monitoramento do transporte de cargas. Esse grupo

é composto por oito classes (*Carga*, *Viagem*, *Transporte*, *Navio*, *Caminhao*, *Alerta*, *Porto* e *CentroDistribuicao*).

O segundo grupo é composto pelas classes de integração, ou seja, são as classes responsáveis por adaptar as funcionalidades FrameTrail ao domínio de aplicação do CargoTrail. Esse grupo é composto por doze classes, que podem ser divididos pelos conceitos do FrameTrail, portanto elas são compostas por três checkpoints (*ViagemCheckPoint*, *AlertaCheckPoint* e *CercaEletronicaCheckPoint*), quatro contextos (*ViagemContext*, *CercaEletronicaContext*, *PortoContext* e *CDContext*), um restrições (*AlertaRestriction*), três eventos (*CercaEletronicaEvent*, *ViagemEvent* e *AlertaEvent*), dois provedor de contexto (*ViagemContextProvider* e *CargoContextProvider*) e três sensores (*PesoSensor*, *InclinacaoSensor* e *TemperaturaSensor*).

O FrameTrail pode ser considerado um framework passível de customização, pois neste estudo de caso não foi necessário alterar nenhuma de suas classes internas para o desenvolvimento do CargoTrail. A integração entre a aplicação e o *framework* foi feita através da especialização dos conceitos do FrameTrail pelo CargoTrail.

6.4 Considerações sobre o capítulo

Este capítulo descreveu a metodologia de avaliação do FrameTrail, assim como os resultados obtidos através dela. Além disso, foram apresentados os quesitos avaliados em cada estudo de caso, assim como os dois protótipos de aplicações desenvolvidos com o auxílio do FrameTrail, apresentando a sua modelagem e implementação.

Levando em consideração os resultados obtidos através da análise dos três quesitos definidos na metodologia de avaliação do presente trabalho, podemos afirmar que o FrameTrail possibilita a construção de aplicações em diferentes tipos de arquitetura e com diferentes tipos de persistência para o armazenamento dos dados de trilhas.

Quanto ao quesito de customização, para os estudos de caso avaliados, o FrameTrail se mostrou customizável e atendeu os requisitos das duas aplicações desenvolvidas. Mas para afirmar que o *framework* é realmente customizável, será necessário que um número maior de desenvolvedores faça uso dele, assim como novas aplicações de diferentes domínios sejam criadas.

O próximo capítulo tem como objetivo dar um fechamento ao trabalho aqui apresentado, destacando suas principais contribuições, bem como sugestões de trabalhos futuros.

7. CONSIDERAÇÕES FINAIS

Este capítulo apresenta uma reflexão sobre as principais contribuições proporcionadas pelo FrameTrail. Completando tal discussão ainda são exibidos trabalhos futuros.

7.1 Contribuições

Este trabalho apresentou o FrameTrail, um *framework* para o desenvolvimento de aplicações orientadas a trilhas. Para a modelagem do FrameTrail, foram analisados cinco projetos que se propõem a desenvolver o mesmo tipo de aplicação, dando foco a sua arquitetura e funcionamento. Para salientar as diferenças entre o FrameTrail e os trabalhos relacionados, é apresentada a Tabela 2, que consiste na Tabela 1 com o acréscimo do FrameTrail.

	Característica	Hermes	TrailTRECer	Startrack	Nidaros	Ubitrail	Frametrail
1	Considera informações de contexto	Sim	Sim	Não	Sim	Sim	Sim
2	Pode ser estendido para outros domínios ou tipos de aplicação	Sim	Não	Sim	Não	Sim	Sim
3	Oferece uma biblioteca destinada ao desenvolvimento de aplicações	Sim	Sim	Sim	Sim	Não	Sim
4	Arquitetura do modelo	Cliente-Servidor	Cliente-Servidor	Cliente-Servidor	Cliente-Servidor	Cliente-Servidor	Cliente-Servidor e P2P
5	Mantém informações históricas	Não	Sim	Sim	Sim	Sim	Sim
6	Local para armazenamento dos dados	Remoto	Remoto	Remoto e Local	Remoto	Remoto	Remoto e Local
7	Oferece suporte a mais	Sim	Não	Sim	Sim	Sim	Sim

	de um tipo de tecnologia de localização						

Tabela 2. Comparativo entre o FrameTrail e demais trabalhos relacionados estudados

O modelo do FrameTrail se propõe a fazer o gerenciamento das informações de contexto através do suporte a componentes externos que podem ser integrados a sua arquitetura através dos provedores de contexto. Esses provedores são responsáveis por consultarem as informações dos mais diferentes tipos de dispositivos e converterem elas para que o *framework* possa gerenciar seus contextos de maneira transparente e integrada, independente do tipo de informação que está sendo analisada.

Quanto ao seu domínio de aplicação, o FrameTrail é direcionado ao desenvolvimento de aplicações orientadas a trilhas, mas essas trilhas não definem um domínio específico, pois o seu foco está voltado para prover os componentes e serviços necessários para que qualquer tipo de aplicação possa obter os benefícios do gerenciamento de trilhas. Sendo assim, fica a cargo de cada aplicação definir o seu domínio sem se preocupar com as possíveis restrições do *framework*.

Como o FrameTrail é um *framework*, sua distribuição é feita através de bibliotecas que podem ser usadas por qualquer tipo de aplicação. A arquitetura de trabalho das aplicações desenvolvidas a partir do FrameTrail é aberta à definição de cada aplicação, ou seja, elas podem seguir em um modelo mais clássico, como o cliente-servidor, ou também elas podem ser executadas usando apenas o dispositivo móvel em questão. Essa definição de arquitetura acontece através da parametrização dos provedores externos, pois eles é que são responsáveis pelo gerenciamento das informações, sejam a partir do computador local ou a através da interação com um servidor de dados.

Quanto ao armazenamento dos dados, a estratégia proposta pelo FrameTrail é a mesma utilizada para o gerenciamento do contexto, ou seja, será o provedor de dados que definirá aonde os dados da trilha serão salvos. O FrameTrail trata as

informações históricas das trilhas de cada entidade através da criação de uma trilha única por entidade, independente de onde e como os dados serão salvos.

O aspecto mais importantes do FrameTrail é a possibilidade de aplicações diferentes que podem ser desenvolvidas através do seu uso. Como visto anteriormente, os trabalhos estudados na área de *frameworks* orientados a trilhas são dependentes de uma infraestrutura de servidores para o seu funcionamento. Com o uso do FrameTrail não existe esse tipo de dependência, pois as aplicações desenvolvidas a partir dele podem tanto executar sem interação com nenhum outro tipo de dispositivo, como podem também acessar um servidor central, assim como também pode ser totalmente descentralizada comunicando-se diretamente com outros dispositivos.

7.2 Conclusões

A computação móvel esta cada vez mais presente no dia a dia das pessoas, não só pela popularização dos dispositivos móveis, mas também pela grande gama de redes sem fio disponíveis atualmente. Com base nisso, o usuário pode executar diversas atividades nas mais diferentes localidades, fazendo uso de vários dispositivos computacionais. Dentro dessa abordagem, as informações referentes à localidade do usuário, assim como as tarefas que ele desempenha, não são devidamente exploradas pelos aplicativos desenvolvidos atualmente, pois a implementação dos controles necessários para tal se torna muito trabalhosa, devido à falta de um padrão entre as aplicações desenvolvidas até o presente momento.

Sendo assim, o FrameTrail foi pensando para padronizar o acesso as informações referentes as trilhas de uma entidade, explorando assim sua total potencialidade. E para que possa ser possível utilizar este trabalho no desenvolvimento de aplicações, optou-se por concebê-lo através de uma das técnicas mais utilizadas de reuso de software: um framework.

Os estudos de caso apresentados no processo de avaliação do FrameTrail permitiram que os elementos definidos no modelo pudessem ser explorados e validados. Dentre esses elementos, o principal deles foi à obtenção de trilhas sensíveis ao contexto em aplicações com domínios bem distintos. Embora a geração dos dados de trilhas para esses estudos de caso tenham sido geradas de forma simulada, os resultados obtidos na validação atenderam as expectativas com relação a padronização da maneira de trabalho com as trilhas, assim como na flexibilidade do armazenamento de suas informações.

7.3 Trabalhos Futuros

Um *framework* normalmente não está totalmente finalizado em suas primeiras versões. Segundo (Pree, 2000), um *framework* precisa ser especializado diversas vezes, continuamente, com o objetivo de detectar falhas, incrementar suas funcionalidades e ainda identificar novos *hot-spots*, que em um primeiro momento não foram definidos. Nesse sentido, é interessante que novas aplicações, de diferentes domínios, sejam desenvolvidas fazendo uso do FrameTrail com o intuito de aprimorar o *framework*.

Com a possibilidade da utilização de provedores externos durante o seu funcionamento, o FrameTrail abre espaço para diversos trabalhos futuros. Dentre eles podemos destacar a implementação de novos provedores de dados, além de novos provedores de comunicação. Atualmente o FrameTrail trabalha com um número limitado de provedores externos. A implementação de novos provedores agregaria novas funcionalidades ao *framework*, fazendo com que ele possa atender as necessidades dos mais variados tipos de aplicação.

Outra área de extensão interessante para o FrameTrail seria a implementação de mecanismo para a identificação de semelhança de trechos entre trilhas de diferentes entidades. Com base nesse mecanismo seria possível relacionar entidades e contextos baseados no seu histórico. Sendo assim, também seria possível criar um mecanismo de inferência para complementar o processo de sugestão de novos

relacionamentos entre as entidades, com base nas informações de diferentes aplicações que façam uso do FrameTrail.

Por fim seria interessante criar a possibilidade de parametrização dos provedores de contexto através de arquivos de configuração externos, onde os mesmos supririam a necessidade de implementação de classes especificas para a vinculação de provedores externos com os contextos gerados pelas aplicações que fazem uso do *framework*.

7.4 Limitações

A grande limitação do presente trabalho é o seu reduzido número de estudos de caso. Como para o processo de avaliação foram desenvolvidas apenas duas aplicações diferentes, por mais que elas explorem as funcionalidades pretendias pelo *framework* proposto, as conclusões aplicam-se apenas a esse universo pequeno de possibilidades.

Outra limitação encontrada no durante o desenvolvimento do FrameTrail foi quanto às tecnologias envolvidas. Por mais que o FrameTrail tenha sido projetado para a sua parametrização visando diferentes tipos de sensores das mais diversas tecnologias, no presente trabalho esse comportamento foi simulado, visando maior agilidade na coleta dos dados para análise.

REFERÊNCIAS BIBLIOGRÁFICAS

ABOWD, G. D; MYNATT, E. D. Charting Past, Present and Future Research in Ubiquitous Computing. ACM Transaction on Computer-Human Interaction. v. 7, n. 1, p. 29-58, Mar. 2000.

ANANTHANARAYANAN et al., "Startrack: A Framework for Enabling Track-Based Applications," Proc. 7th Int'l Conf. Mobile Systems, Applications, and Services (MobiSys 09), ACM Press, 2009, pp. 207–220.

BARBOSA Jorge, HAHN, Rodrigo, RABELLO, Solon, BARBOSA, Débora LOCAL: a Model Geared Towards Ubiquitous Learning. Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education, SIGCSE, 2008.

BUSH, Vannevar. As We May Think. The Atlantic Monthly. 1945.

CHEN, G.; KOTZ, D. A Survey of Context-aware Mobile Computing Research. USA: Dartmouth Computer Science, 2000. (Technical Report, TR2000-381). Disponível em: < http://citeseer.ist.psu.edu/chen00survey.html >. Acesso em: mar. 2010.

CHEN, G., KOTZ, D. Context Aggregation and Dissemination in Ubiquitous Computing Systems. In: IEEE WORKSHOP ON MOBILE COMPUTING SYSTEMS AND APPLICATIONS, WMCSA, 4., 2002. Proceedings… Calligon, New York: IEEE Computer Society Press, 2002. p.105-114.

CRESPO, S. (2000). Composição em WebFrameworks. Phd thesis in computer science, PUC-Rio.

DEY, A.K., et al. 1999. Towards a better understanding of context and contextawareness. In Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing. Springer-Verlag, 304-307.

DRIVER, C. 2007. An Application Framework for Mobile, Context-Aware Trails. PhD thesis, University of Dublin.

EDWARDS, W. K., Bellotti, V., Dey, A. K., and Newman, M. W. (2003). The challenges of user-centered design and evaluation for infrastructure. pages 297–304, Ft. Lauderdale, Florida, USA. ACM.

FAYAD, M. E., Schmidt, D. C., and Johnson, R. E. (1999). Building application frameworks: object-oriented foundations of framework design. New York: John Wiley & Sons.

FONTOURA, M. F. M. C. (1999). A Systematic Approach for Framework Development. Phd thesis in computer science, PUC-Rio.

GAMS, E., and Reich, S. The TrailTRECer framework: Applying open hypermedia concepts to trails. Journal of Universal Computer Science 8, 10 (2002), 913 – 923.

HENRICKSEN, K., INDULSKA, J. A Software Engineering Framework for Context-Aware Pervasive Computing, In: IEEE INTERNATIONAL CONFERENCE ON PERVASIVE COMPUTING AND COMMUNICATIONS, PERCOM, 2., Orlando, 2004. Proceedings... Los Alamitos, CA: IEEE Computer Society, 2004. p. 77-86.

HIGHTOWER, J. and Gaetano, B. 2001. Location Systems for Ubiquitous Computing. IEEE Journal, IEEE Press, 34, 8, 57-66.

IEEE standard for information part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. http://standards.ieee.org/getieee802/download/802.11-2007.pdf. Acesso em: março de 2010.

IMT - International Mobile Telecommunications-2000 (http://www.itu.int/home/ imt.html. Acesso em: março de 2010.

LAMARCA Anthony, Yatin Chawathe, Sunny Consolvo, Jeffrey Hightower, Ian E. Smith, James Scott, Timothy Sohn, James Howard, Jeff Hughes, Fred Potter, Jason Tabert, Pauline Powledge, Gaetano Borriello, and Bill N. Schilit. Place lab: Device positioning using radio beacons in the wild. In Pervasive, pages 116–133, 2005.

LEVENE, Mark; PETERSON, Don. Trail Record and Ampliative Learning. Research Report BBKCS-02-01, School of Computer Science and Information Systems. University of London, 2002.

MATTSSON, M. (1996). Object-oriented frameworks - a survey of methodological issues.

MOBILEIN Technologies. Location Based Services (LBS), Acessado em novembro de 2009 [Disponível em http://www.mobilein.com/location_based_services.htm]

PARAMVIR Bahl and Venkata N. Padmanabhan. Radar: An inbuilding rf-based user location and tracking system. In INFOCOM, pages 775–784, 2000.

PREE,W. (2000). Hot-spot-driven framework development. In Fayad,M., Schmidt, D., and Johnson, R., editors, Building Application Frameworks: Object-Oriented Foundations of Framework Design. Wiley & Sons, New York City.

RANGANATHAN, A., CAMPBELL, R. H. A Middleware for Context-Aware Agents in Ubiquitous Computing Environments. In: INTERNATIONAL MIDDLEWARE CONFERENCE, 2003. Proceedings… Rio de Janeiro, ACM Press, p. 143-161.

SACRAMENTO, Vagner et al. MoCA: A Middleware for Developing Collaborative Applications for Mobile Users. IEEE Distributed Systems Online. v. 5, n. 10, Oct. 2004.

SATYANARAYANAN, M. Fundamental Challenges in Mobile Computing. In: ACM SYMPOSIUM ON PRINCIPLES OF DISTRIBUTED COMPUTING, 1996. Proceedings... Philadelphia, Pennsylvania, USA: Berlin: Springer-Verlag, 1996.

SCHMIDT, D., Stal, M., Rohnert, H., and Buschmann, F. (2001). Pattern-oriented software architecture : patterns for concurrent and networked objects. Chichester: John Wiley & Sons, 1st edition.

SILVA, Jader Marques. UbiTrail: Um Modelo para Gerenciamento de Trilhas Orientado a Ambientes Ubíquos. Unisinos, São Leopoldo –RS Prop. Dissertação, Junho 2009.

SMITH, A. 2008. Who Controls the Past Controls the Future - Life Annotation in Principle and Practice. PhD thesis, University of Southampton.

SUN Microsystems, Inc. – Java Technology, 2010. Disponível em: http://java.sun.com/

VELJO Otsason, Alex Varshavsky, Anthony LaMarca, and Eyal de Lara. Accurate gsm indoor localization. In Ubicomp, pages 141–158, 2005.

WANG Alf Inge, Carl-Fredrik Sørensen, Steinar Brede, Hege Servold, Sigurd Gimre: "The Nidaros Framework for Development of Location-aware Applications", IFIP TC8 Working Conference on Mobile Information Systems - 2005 (MOBIS), Leeds, UK, December 6-7, 2005, 15 pages.