UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

VICTOR YOSHIAKI MIYAI

Desenvolvimento de um Repositório de Dados Unificado para Sistemas Móveis

Trabalho de Graduação.

Prof. Dr. Lucinéia Heloisa Thom
Orientador

Prof. Dr. Bernhard Mitschang
Co-orientador

Porto Alegre, julho de 2012.

AGRADECIMENTOS

Agradeço a minha família e meus amigos pela motivação que deram durante os tempos mais difíceis do curso e da execução deste trabalho. Agradeço aos professores e funcionários, que deram as condições para que eu pudesse escrever este trabalho da melhor forma dentro do possível.

Agradeço o professor Bernhard Mitschang, pessoa que possibilitou o programa de intercâmbio na Universidade de Stuttgart, onde trabalhei no projeto que se tornou as bases deste trabalho. Agradeço também Andreas Brodt, por permitir que eu trabalhasse junto ao projeto desenvolvido para sua tese de doutorado. Essas experiências permitiram a expansão da minha visão sobre o mundo e auxiliou a determinar com maior firmeza minhas ambições daqui para frente.

Finalmente, gostaria de agradecer a banca examinadora deste trabalho, pela paciência de avaliar este trabalho, e minha orientadora, Lucinéia Thom, pela paciência de orientar este aluno dentro do possível.

Aproveito a oportunidade para pedir desculpas a todos que de alguma forma acabei prejudicando durante a execução deste trabalho.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

UFRGS	Universidade Federal do Rio Grande do Sul
IEEE	Institute of Electrical and Electronics Engineers
W3C	World Wide Web Consortium
SMS	Short Message Service
GPS	Global Positioning System
DCCI	Delivery Context: Client Interfaces
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
SSL	Secure Sockets Layer
WSDL	Web Services Definition Language
SOAP	Simple Object Access Protocol
REST	Representational State Transfer
JSON	JavaScript Object Notation
XML	Extensible Markup Language
CRUD	Create, Read, Update, Delete
API	Application Programming Interface
XPCOM	Cross Platform Component Object Model
URI	Uniform Resource Identifier
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
FOAF	Friend of a Friend
SPARQL	SPARQL Protocol and RDF Query Language
SPARUL	SPARQL/Update
MB	Megabyte
kB	Kilobyte
SQL	Structured Query Language

LISTA DE FIGURAS

RESUMO

Este trabalho apresenta uma proposta de solução para os problemas provocados pela redundância e a inconsistência de dados em dispositivos móveis através do uso de um repositório de dados unificado no modelo RDF, oferecendo interoperabilidade a nível de dados para aplicações Web e para aplicações nativas para dispositivos móveis.

Aparelhos celulares e outros dispositivos possuem aplicações que frequentemente trabalham sobre domínios de dados semelhantes aos trabalhados por aplicações na Internet (por exemplo, aplicações envolvendo mapas). Isso provoca a ocorrência de dados redundantes nos repositórios de dados privados de cada aplicação e, com o tempo, inconsistências, porque aplicações geralmente não possuem conhecimento da existência dos mesmos dados nos repositórios privados de outras aplicações, o que impede as aplicações de saberem que uma das cópias dos dados foi atualizada por outra aplicação.

A parte prática do trabalho busca solucionar os problemas implementando parcialmente uma arquitetura de sistema proposta por Brodt et al. (2011), baseado em um repositório de dados unificado trabalhando sobre o modelo de dados RDF onde aplicações podem descobrir e utilizar informação criada por outras aplicações, evitando quando possível o uso de repositórios de dados privados.

Por ser um mecanismo desenvolvido para ser compreendido por máquinas, o modelo RDF permite que aplicações compartilhem informação de forma implícita (sem um protocolo de comunicação explícito), fazendo com que eventuais cópias de dados redundantes em um repositório remoto possam ser atualizadas (retornando a um estado consistente) consultando o repositório unificado de forma automática e sem a intervenção do usuário.

Apesar de apresentar os resultados esperados de interoperabilidade entre aplicações, este trabalho conclui com a necessidade de elaboração de um estudo de caso com aplicações mais complexas baseadas em um cenário real para validar o potencial da solução com maior grau de certeza.

Palavras-Chave: repositório de dados unificado, RDF, Resource Description Framework, redundância de dados, inconsistência, dispositivos móveis, celulares, aplicações, Web semântica.

Development of a Unified Data Repository for Mobile Systems

ABSTRACT

This work presents a solution for problems caused by data redundancy and inconsistency in mobile devices based on an unified data repository working on data modeled in RDF, offering data interoperability between Web and mobile devices applications.

Cell phones and other mobile devices frequently deal with similar data domains like the ones which Web applications often work (as an example, map applications). This causes the existence of multiple copies of redundant data inside private data silos, which evolve to inconsistencies over time, because applications usually have no idea that the same data is already stored in some other application's data repository. Therefore, applications cannot know that a copy of redundant data was updated by another application, leading to inconsistencies.

The practical part of this work tries to solve the problems of data redundancy and inconsistency by partially implementing the system architecture proposed by Brodt et al. (2011), which is based on an unified data repository working with RDF data that allows applications do discover and use data created by other applications, avoiding when possible the use of private data silos.

As a mechanism developed to be understood by machines, the RDF model allows applications to share data in an implicit way (without the use of an explicit communication protocol). This enables applications to update private copies of redundant data without the user's direction by automatically querying the unified data repository and checking for inconsistencies.

Although the implementation has obtained sufficient results to enable interoperability, this work concludes with the need for a case study with more complex applications, based on real life scenarios, to validate the potential of the solution with more confidence.

.

Keywords: unified data repository, RDF, Resource Description Framework, data redundancy, inconsistency, mobile devices, cell phones, applications, semantic Web.

1 INTRODUÇÃO

1.1 Contexto

Aparelhos celulares e outros dispositivos móveis pessoais têm adquirido usuários em um ritmo acelerado: centenas de milhares desses dispositivos são ativados todos os dias e novos usos são descobertos em grande velocidade. Conforme a sociedade atual precisa estar acessível e disponível a praticamente todo o momento, dispositivos móveis têm ganhado um número crescente de funcionalidades e cada vez mais aplicações são desenvolvidas para permitir que usuários personalizem seus dispositivos da forma que lhes são mais convenientes.

Por serem convenientes e facilmente portáveis, as pessoas praticamente vivem juntas de seus aparelhos e, por estarem juntos das pessoas, esses dispositivos vêm desempenhando cada vez mais funções que são tradicionalmente realizadas pelo navegador ou pelo computador pessoal. Em repositórios de aplicações para dispositivos móveis (AppStore, Google Play, etc), é possível encontrar clientes de serviços oferecidos pela Web, como redes sociais, mensageiros instantâneos e jogos casuais, assim como já existem aplicações que são simplificações de software usado em computadores pessoais, como clientes de e-mail e editores de documentos.

Sendo a oferta de aplicações alta, o usuário possui liberdade para escolher uma aplicação que seja mais interessante de ser usada em um dado momento. Como exemplo, o usuário pode utilizar arbitrariamente um navegador de GPS com suporte a mapas offline em seu dispositivo móvel assim como um serviço de mapas online na Web enquanto está utilizando seu computador pessoal em casa. Como as duas aplicações trabalham sobre domínios de dados semelhantes (pontos de interesse e posições geográficas) e que nesse tipo de situação, sobreposição de dados ocorre com relativa facilidade (ex. um ponto de um trajeto marcado no navegador GPS e o mesmo local marcado como favorito na aplicação de mapa), as duas aplicações armazenam uma cópia da informação relativa ao mesmo ponto de interesse em seus repositórios privados.

Conforme o usuário usa diversas aplicações de domínios sobrepostos indiscriminadamente, cópias de dados representando a mesma entidade semântica são adicionadas em repositórios privados (locais ou remotos), produzindo redundância de informação.

Aplicações em geral dificilmente possuem funcionalidades que permitem expor seus dados privados para uso de outras aplicações, assim como raramente possuem funcionalidades para procurar informação disponível nos repositórios privados de outras aplicações. Adicionalmente, não existem garantias que aplicações de mesmo domínio

utilizam os mesmos modelos de dados para um mesmo elemento ou entidade (ex. coordenadas geográficas representadas como uma única *string* “lat:x,long:y” em uma aplicação e como um par de *strings* {lat:”x”, long:”y”} em outra). Essas razões tornam difícil a troca de dados entre aplicações: caso uma aplicação receba uma atualização dos dados redundantes (ex. mudança de endereço do usuário), ela fará atualização apenas na sua cópia privada dos dados, enquanto que as outras cópias permanecem no seus estados originais. Caso o usuário não atualize as informações das outras aplicações, os dados estarão em um estado inconsistente.

1.2 Motivação

Sintetizando o que foi dito anteriormente, o uso arbitrário de aplicações de domínios relacionados implica em múltiplas cópias de dados que representam uma mesma entidade semântica, ou seja, redundância de dados. Conforme aplicações são geralmente incapazes de comunicar mudanças nesses dados redundantes, inconsistências tendem a surgir com o passar do tempo.

Redundância de dados tende a prejudicar a vida de usuários porque é de responsabilidade dos mesmos manter as próprias informações atualizadas: sem ferramentas que sejam capazes de fazerem alterações automatizadas, um usuário precisaria alterar, por exemplo, um campo de endereço comercial em todos os serviços e aplicações que exibem ou usam esse endereço, manualmente. Já a inconsistência de dados causa transtornos quando pessoas ou aplicações utilizam esses dados inconsistentes: ainda usando o exemplo de endereço, um cliente que visitasse o endereço comercial de um profissional ficaria bastante frustrado ao descobrir que o endereço apontado pela homepage do dito profissional (ou por um catálogo de serviços online) estava desatualizado.

Considerando que uma parcela apreciável das ocorrências de inconsistência de dados são decorrentes da redundância dos mesmos, uma solução que reduzisse a quantidade de cópias privadas de dados em um conjunto de aplicações de domínios relacionados contribuiria diretamente na redução de dados redundantes e, indiretamente, na diminuição de inconsistências. Mesmo sem eliminar cópias, uma solução que pudesse fazer o gerenciamento de dados redundantes de forma controlada também seria uma solução interessante para gerenciar inconsistências.

Em suma, uma solução que resolva os problemas de redundância e inconsistência de dados, mesmo que parcialmente, possibilitaria uma forma de gerenciar informação de forma mais inteligente.

1.3 Proposta

Este trabalho descreve a implementação de um protótipo de sistema com funcionalidades que buscam atenuar os problemas provocados pela redundância e inconsistência de dados em dispositivos móveis através do alcance da interoperabilidade a nível de dados entre aplicações. Essa interoperabilidade é alcançada através do uso de um repositório de dados unificado que armazena dados criados por uma aplicação para serem utilizados também por outras aplicações. Ao centralizar escritas e leituras de informação em um único repositório, aplicações podem ter uma visão global dos dados armazenados e assim evitar a gravação de cópias de dados já existentes.

Para que aplicações consigam trabalhar sobre os mesmos dados, é necessário que o repositório de dados unificado e as aplicações que operam sobre ele usem um modelo de dados padrão. Esse modelo de dados deve ser flexível para permitir que informações criadas por uma aplicação sejam enriquecidas com informações adicionais criadas por outras aplicações, sem afetar a sua compreensão.

Adicionalmente, o sistema deve ser capaz de lidar com a existência de cópias de dados redundantes de forma controlada, como ocorre em aplicações onde cópias locais são necessárias, tipicamente serviços oferecidos pela Internet. Para isso, é necessário que o repositório seja capaz de oferecer interoperabilidade tanto para aplicações nativas do dispositivo como para aplicações na Web.

Este trabalho tem suas bases em um trabalho desenvolvido em colaboração com o Instituto de Sistemas Paralelos e Distribuídos da Universidade de Stuttgart sob a orientação do professor Bernhard Mitschang como um projeto relacionado à tese de doutorado de Andreas Brodt. O sistema desenvolvido para este trabalho utiliza os conceitos da arquitetura descrita por Brodt et al. (2011), começando originalmente como um protótipo funcional de um repositório de recursos e dados de contexto (com ênfase em dados de posição geográfica) e terminando como uma implementação diferente que tem como objetivo diminuir os efeitos negativos causados pela redundância e pela inconsistência de dados.

1.4 Trabalhos Relacionados

Este trabalho é fortemente baseado na arquitetura proposta por Brodt et al. (2011), que propõe uma arquitetura para dispositivos móveis que fornece interoperabilidade entre aplicações nativas do dispositivo, aplicações Web, dados originados de sensores do aparelho e outros dispositivos próximos a nível de dados.

Outro trabalho relacionado a este é a extensão para o navegador Mozilla Firefox, *SparqlForwarder*, desenvolvido como parte da tese de doutorado de Andreas Brodt, da Universidade de Stuttgart. Essa extensão serviu de base para o desenvolvimento da implementação proposta no capítulo 4 deste trabalho.

Um projeto relacionado aos dois trabalhos mencionados anteriormente, Telar DCCI, é uma extensão de navegador que disponibiliza as leituras do sensor de GPS interno a um dispositivo para ser utilizado por páginas Web no navegador MicroB (BRODT e NICKLAS, 2008). Esse projeto pode ser visto como o precursor da ideia do uso de um repositório de dados unificado.

Este trabalho está fortemente relacionado à iniciativa de Web semântica na parte de modelo de dados, apresentando um exemplo de aplicação que utiliza o modelo de dados RDF (usado na Web semântica) em uma aplicação nativa para dispositivos móveis.

1.5 Contribuição

A contribuição deste trabalho é a demonstração de uma forma de reduzir problemas de redundância e inconsistência de dados, utilizando um conceito de interoperabilidade a nível de dados entre aplicações, fora da abordagem mais comum de armazenamento de dados pessoais em servidores remotos ou repositórios locais com trocas de dados realizadas através de um protocolo de comunicação definido entre as aplicações.

Este trabalho possui o potencial de influenciar a forma como informação pode ser armazenada e consultada por aplicações de forma mais inteligente e como aplicações podem utilizar dados criados por outras aplicações para oferecer um serviço de maior qualidade ou realizar uma tarefa com maior precisão para o usuário.

Adicionalmente, este trabalho apresenta uma forma de como os conceitos de Web semântica podem ser utilizados para a solução de problemas que vão além do contexto da Web.

1.6 Organização do Trabalho

Este trabalho está organizado da seguinte forma: o segundo capítulo dá introdução aos conceitos fundamentais ao entendimento deste trabalho. O terceiro capítulo detalha a arquitetura proposta por Brodt et al., no qual este trabalho é fortemente baseado. O quarto capítulo apresenta os detalhes de implementação do protótipo de um repositório unificado. O quinto e último capítulo apresenta o fechamento deste trabalho, levantando as conclusões do trabalho, as dificuldades encontradas e finalmente, os trabalhos futuros.

2 CONCEITOS

Este capítulo introduz conceitos que são essenciais à compreensão das próximas seções, apresentando aspectos de tecnologias relacionadas com a arquitetura que será apresentada no capítulo 3 e com a proposta de implementação de um protótipo de repositório de dados unificado, apresentada no capítulo 4. Considerando a abrangência de áreas envolvidas neste trabalho, os tópicos serão abordados com o uso extensivo de exemplos para facilitar a compreensão.

Primeiramente, são abordados neste capítulo noções de interoperabilidade entre aplicações, de forma a justificar uma solução baseada em um repositório de dados unificados. Em seguida, é dada uma visão geral sobre como aplicações Web comunicam entre si e com o navegador. Levando em conta que um dos objetivos secundários deste trabalho é a obtenção de interoperabilidade entre aplicações, é importante apresentar as formas como aplicações Web podem armazenar e trocar informação.

Depois, é dada uma introdução sobre o movimento da Web semântica e sua relação com o Resource Description Framework, modelo de dados utilizado no repositório da parte de implementação que oferece uma forma flexível de criar relações entre dados em uma linguagem compreensível por aplicações.

Por fim, alguns conceitos da plataforma Android, da Google, são apresentados de forma a permitir a compreensão dos detalhes de desenvolvimento da proposta de implementação, focando na forma como armazenamento de dados pode ser realizada e como aplicações interagem entre si.

2.1 Interoperabilidade

Interoperabilidade é definida pela IEEE como “a habilidade de dois ou mais sistemas ou componentes trocarem informações e utilizarem as informações trocadas” (IEEE, 1990). Uma aplicação pode ser definida como software projetado para auxiliar usuários a realizarem uma tarefa específica. O termo aplicação é usado extensivamente neste trabalho e ocasionalmente o termo “aplicativo” é usado como sinônimo de uma aplicação para dispositivos móveis. O termo “aplicação Web” também é frequentemente usado como sinônimo do termo “página Web”.

Aplicações em geral não possuem funcionalidades que permitem trocas de dados por uma grande variedade de motivos: falta de padrões para desenvolvimento de software, protocolos de comunicação incompatíveis, modelos de dados incompatíveis, aumento do tempo de desenvolvimento para a inclusão de funcionalidades para troca de dados, aumento da complexidade da aplicação, diferenças entre plataformas e assim por diante. Estes exemplos são apenas uma fração dos motivos existentes que impedem

interoperabilidade efetiva entre aplicações e não é objetivo deste trabalho enunciar, tampouco analisar todos problemas existentes.

Por motivos como esses, aplicações que criam e gerenciam os próprios dados são muito mais simples de serem desenvolvidas. Como consequência, aplicações que armazenam informação em repositórios de dados privados são muito mais frequentes, provocando os problemas já conhecidos de redundância e inconsistência de dados.

2.1.1 Vantagens da Interoperabilidade em Dispositivos Móveis

Em dispositivos móveis, aplicações frequentemente trabalham sobre os mesmos domínios de dados, gerando grandes quantidades de dados redundantes que poderiam ser compartilhados entre múltiplas aplicações. Aplicações interoperáveis seriam beneficiadas pela existência de dados criados por outras aplicações porque esses dados adicionais podem ser usados para:

- Oferecer conteúdo mais rico: anúncios de publicidade online frequentemente usam dados obtidos através de *trackers* registrados como *cookies* HTTP para analisar os hábitos de navegação do usuário e assim apresentar propaganda com maior chance de interessar o usuário. Aplicações como páginas Web poderiam acessar dados gerados por outras aplicações para adaptar seu conteúdo e torná-lo mais interessante com base nas preferências do usuário
- Automatizar tarefas: se aplicações interoperáveis trabalham sobre os mesmos domínios de dados, elas podem compartilhar os mesmos recursos, fazendo com que atualizações em dados sejam propagadas automaticamente para todas as aplicações que utilizam esses dados específicos. Adicionalmente, aplicações que necessitam cópias de dados em repositórios privados (como serviços oferecidos pela Internet) podem facilmente localizar dados inconsistentes em seus repositórios (comparando-os com os dados de outras aplicações) e atualizá-los de forma automática, reduzindo a necessidade da interação do usuário e a quantidade de esforço necessária para a realização de tarefas repetitivas
- Realizar tarefas com maior acurácia: aplicações em geral são incapazes de realizar tarefas sem o julgamento e a intervenção do usuário: uma busca pela palavra "die" em ferramentas de busca *online* poderiam retornar ao usuário registros sobre o verbo em inglês "morrer" ("*to die*") ou o substantivo em inglês para "dado" ("*die*", plural: "*dice*"), quando o usuário estava, na verdade, procurando pelo artigo definido feminino em alemão ("*die*"). Se aplicações têm acesso a dados de contexto como posição geográfica do usuário ou eventos de calendário criados por outras aplicações, uma busca por "*die* alemão" poderia gerar resultados diferentes se o usuário está em casa e existe um registro no calendário contendo um evento com significado "fazer tema de alemão" ou se o usuário está em Bonn, na Alemanha e possui um compromisso registrado no calendário com o significado de "reunião no DIE" (DIE - Deutsches Institut für Entwicklungspolitik, Instituto Alemão de Desenvolvimento).

2.1.2 Repositório de Dados Unificado

Para se obter interoperabilidade de dados entre aplicações diversas, os repositórios de dados privados de cada aplicação podem possuir funcionalidades que permitem a troca de dados entre si. Porém, como visto no início deste capítulo, a implementação

dessas funcionalidades normalmente implica no aumento da complexidade, dos custos e do tempo de desenvolvimento de aplicações.

Uma opção possível mais simples é o uso de um único repositório de dados utilizado por todas as aplicações, substituindo parcial ou totalmente um banco de dados privado. Esse repositório de dados unificado basicamente centraliza as operações de manipulação de dados das aplicações em um único banco de dados, onde informação criada por uma aplicação pode ser, em teoria, lida e escrita por outras aplicações. Na prática, mecanismos de segurança internos regulam quais aplicações possuem acesso a um determinado dado ou classe de dados.

Um repositório unificado pode expor uma interface às aplicações que fazem uso dele de forma a facilitar operações, semelhante à forma como bancos de dados relacionais internos às aplicações fazem: através da abertura de uma conexão com o repositório e realizando chamadas de operações sobre um objeto de manipulação do banco de dados, resultando em poucas alterações necessárias em uma aplicação para suportar o uso do repositório.

Vantagens oferecidas por um repositório de dados unificado incluem:

- Acesso a dados compartilhados de forma mais simples, uma vez que dados criados por outras aplicações já estão presentes no repositório, dispensando comunicação com repositórios de dados privados
- Redução de dados redundantes, caso o repositório implemente condições para que aplicações consigam encontrar dados que já existem antes de criarem um novo registro. Neste caso, um modelo de dados padrão para as aplicações e a definição explícita de dados já existentes tornam-se necessários
- Presença dos dados localmente em dispositivos móveis, considerando que o repositório seja interno ao dispositivo. Como aplicações nativas do dispositivo dependem de dados no repositório, e aplicações Web só podem acessar os dados do repositório na presença de uma conexão com a Internet, o usuário, dono legítimo das informações armazenadas, tem posse dos dados sem depender de serviços de terceiros.

No entanto, também existem desvantagens que devem ser consideradas:

- Para alcançar interoperabilidade, aplicações e o repositório unificado devem trabalhar sobre o mesmo modelo de dados, ou dados poderão ser lidos, mas não compreendidos. Dados que não podem ser compreendidos por todas as aplicações promove o surgimento de cópias de dados redundantes quando aplicações trabalham sobre domínios semelhantes: apesar de representarem uma mesma entidade semântica, a representação sintática desses dados no repositório é diferente, portanto, considerados entidades diferentes. Nem todos os desenvolvedores podem chegar em um consenso sobre qual modelo de dados utilizarem em suas aplicações
- Necessidade de implementação de mecanismos rígidos de segurança, de forma a identificar uma aplicação que deseja realizar operações sobre o repositório e determinar se essa aplicação pode realmente realizar as operações que está solicitando.

- Propagação de dados incorretos, uma vez que a atualização incorreta de informações realizada por uma aplicação poderá comprometer todas as outras que utilizam essas informações.

Para a proposta de implementação descrita no capítulo 4, um repositório de dados unificado é desenvolvido utilizando um modelo de dados padrão que é utilizado na iniciativa da Web semântica, descrita na seção 2.3. O repositório desenvolvido, no entanto, não possui mecanismos de segurança implementados devido à sua complexidade. A próxima seção descreve em maiores detalhes sobre autenticação de páginas Web e o capítulo 3 descreve o uso de controle de acesso baseado em AccessRoles ("papéis" de acesso), ambos relevantes para a parte de segurança.

2.2 Aplicações Web

2.2.1 Interoperabilidade entre Aplicações Web

Podemos observar basicamente dois tipos de interoperabilidade de dados para aplicações Web. A utilizada por aplicações como Web services é baseada na troca de dados a partir de um protocolo definido, de uma aplicação para outra e em linguagem compreensível por máquinas. A outra é baseada em metadados associados aos dados exibidos em uma página Web para um usuário. Apesar de uma aplicação como um navegador ser capaz de ler a informação enviada por uma página (ex. uma tabela codificada em HTML), ela é normalmente incapaz de compreender a semântica desses dados (ex. uma tabela com preço e descrição de guitarras de uma página Web que vende instrumentos musicais).

Web *services* são um bom exemplo de sistemas que alcançam um nível considerável de interoperabilidade do primeiro tipo, onde aplicações expõem interfaces definidas em linguagens processáveis por máquinas (normalmente em WSDL) e operações são executadas através de mensagens no protocolo SOAP ou de operações padronizadas da arquitetura REST. Características de interoperabilidade de dados em nível semântico são abordadas na seção 2.3 na área de Web semântica.

2.2.2 Navegador

O navegador é basicamente a plataforma onde páginas Web são apresentadas a um usuário. Na questão de interoperabilidade de dados, o navegador é a aplicação nativa que realiza o intermédio entre páginas Web e dados disponíveis nos repositórios de dados internos da plataforma, além de possibilitar a comunicação entre páginas abertas em abas na mesma janela.

No entanto, a maior parte dos navegadores não tem essas funcionalidades habilitadas por causa da grande quantidade de páginas Web que possuem conteúdo malicioso e que poderiam acessar ou modificar dados sem permissão. Navegadores também implementam isolamento de processos de forma semelhante à sistemas operacionais, impedindo que páginas maliciosas influenciem de forma negativa o comportamento de outras páginas em outras abas.

Alguns navegadores como o Mozilla Firefox permitem que o usuário decida se deseja conceder privilégios a uma página Web que solicite operações potencialmente perigosas para a segurança. A figura 2.1 apresenta uma caixa de diálogo gerada pelo navegador na qual uma página deseja instalar ou executar software na plataforma.

Alguns navegadores suportam extensões ou modificações onde o tratamento de solicitações de recursos restritos pode ser feito de forma diferente (através de uma interface gráfica exposta pelo navegador para o usuário). Um exemplo de tal extensão é apresentada brevemente no capítulo 4, sendo o precursor do protótipo desenvolvido para a proposta de implementação deste trabalho.

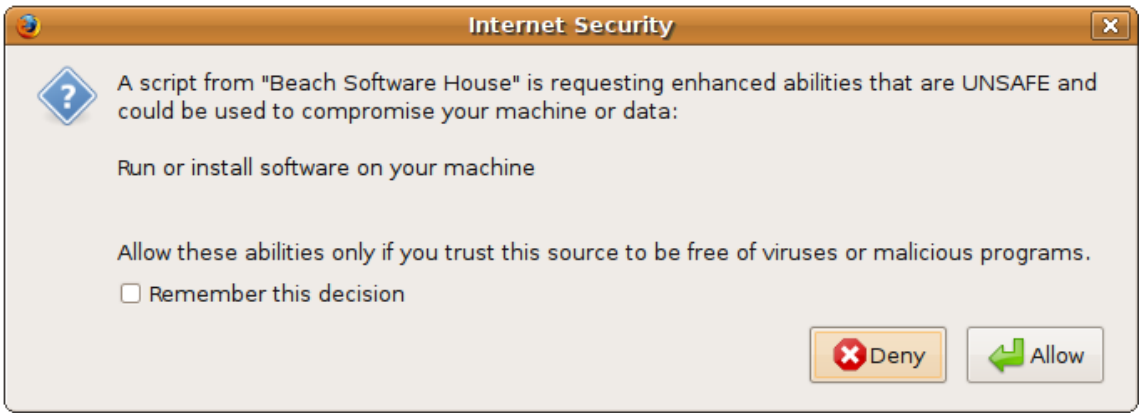

Figura 2.1: Exemplo de diálogo solicitando permissões no Mozilla Firefox.

Em dispositivos móveis, páginas Web acessadas através de navegadores teriam uma grande fonte de dados de contexto (leituras de GPS, lista de chamadas recebidas, entre outros) para enriquecer seus dados e assim apresentar conteúdo mais interessante ao usuário. Essas interações devem ser feitas através do navegador, implicando na implementação de funcionalidades de troca de informação e de um protocolo comum entre as aplicações e o navegador.

2.2.2.1 Armazenamento Local

A forma como a maior parte das páginas Web armazenam informação persistente localmente para uso posterior é através de *cookies* HTTP, frequentemente usados para armazenamento de dados de sessão, personalização de páginas e monitoramento do comportamento do usuário através de pares de chave e valor (*key-value pair*).

Com o surgimento do HTML5, formas alternativas de armazenamento local foram propostas. Entre eles as APIs Web *storage*, que permite o armazenamento de pares de chaves e valores de forma semelhante aos *cookies* HTTP, mas com maior capacidade de armazenamento (5 a 10 MB, contra 4kB dos *cookies*) e *Indexed Database*, de mesmo propósito que o Web *storage* e suportado por diversos navegadores (Google Chrome, Mozilla Firefox, Opera e Safari).

O armazenamento de dados dentro do navegador seria uma forma interessante de unificar um repositório de dados para aplicações junto da plataforma que interage com páginas Web, uma vez que a implementação de funcionalidades para comunicação entre o repositório e o navegador se tornaria desnecessário (o navegador é o repositório).

Porém, navegadores não permitem que outras aplicações Web tenham acesso a dados que não foram criadas por elas mesmas ou por aplicações que não são do mesmo domínio, tanto para *cookies* como para as formas de armazenamento de dados disponíveis no padrão HTML5. Uma alternativa para permitir o compartilhamento de

dados é o desenvolvimento de extensões para navegadores que permitem formas de armazenamento de dados persistentes alternativas. A extensão para o navegador Mozilla Firefox, *SparqlForwarder*, implementa um repositório de dados interno compartilhado entre várias aplicações Web. Mais detalhes sobre a extensão são descritos no capítulo 4.

2.2.3 Autenticação

Para que páginas Web tenham acesso aos dados de um repositório, é necessário que a página seja primeiramente autenticada para evitar que uma aplicação Web realize operações sobre o repositório em nome de outra aplicação (*spoofing*), possivelmente compromentendo a integridade dos dados e a privacidade do usuário.

Autenticação é uma tarefa realizada entre o navegador e uma aplicação Web e é recomendado que seja realizada através do protocolo HTTPS para evitar ataques *man-in-the-middle* durante o envio de requisições de operações sobre um repositório de dados.

2.3 Web Semântica

Web semântica é uma iniciativa da W3C que tem como objetivo tornar a interação entre aplicações Web mais fácil, adicionando semântica aos dados em um formato compreensível por máquinas. Atualmente, a maior parte das páginas Web disponíveis na Internet possuem conteúdo legível tanto por máquinas como por pessoas. No entanto, o conteúdo é apresentado em um formato que é normalmente compreensível apenas por pessoas, o que ocorre devido à capacidade das pessoas de atribuir **significado** ao conteúdo sintático exibido em uma página, bem como estabelecer **relações** entre as entidades representadas por esse conteúdo.

2.3.1 Motivação

Usemos como exemplo uma foto retirada durante um churrasco de aniversário, exibida em uma página Web com nome de arquivo “Meu Aniversário 04-07-2012 - 030”. Para um ser humano, é facilmente compreensível de que se trata de uma foto retirada em um aniversário do proprietário da página que ocorreu no dia 4 de Julho de 2012 e que é a foto de número trinta de um conjunto de fotos. Através da observação da imagem, é possível constatar que o evento foi comemorado com um churrasco. Já para uma aplicação (como um navegador), as informações que ela possui são de que a página possui um elemento de imagem associado ao arquivo de nome “Meu Aniversário 04-07-2012 - 030” e de que essa imagem é apresentada na página em uma determina posição.

Apesar do arquivo poder ter dados extras como “data de criação do arquivo” ou “tamanho de arquivo”, eles não oferecem informações adicionais que permitem que a aplicação conclua que a imagem representa a “trigésima foto do churrasco de aniversário do usuário ocorrido no dia 04 de Julho de 2012”. Mesmo que exista informação adicional, como um *label*, contento exatamente o trecho mencionado, uma aplicação seria incapaz de entendê-la, uma vez que a informação foi feita para ser compreendida por pessoas, e não por máquinas.

2.3.2 Redes de Dados

A Web semântica busca solucionar os problemas como os observados no exemplo através de basicamente duas ideias: estabelecer um formato de dados comum que possa

ser usado para tecer uma rede de significados e integrar aplicações e relacionar dados com entidades do mundo real

Um formato de dados comum para representar conhecimento permite que aplicações possam utilizar informação de múltiplas fontes, de forma a enriquecer os próprios dados e realizar tarefas com maior eficiência, contando com a cooperação de outras aplicações.

A atribuição de significado aos dados e sua relação com o mundo real permite que informações sejam compreendidas e conectadas a outros fragmentos de informação, formando assim uma verdadeira "rede de significados" onde a semântica de um determinado elemento pode ser inferida através da navegação nessa rede: sabendo que a foto do dito aniversário é extendida com dados como data ("04 de Julho de 2012") e participantes da foto ("o próprio usuário"), uma aplicação de calendário com um evento marcado na mesma data ("04 de Julho de 2012") e extendida com informação como tipo de evento ("aniversário"), participantes ("o próprio usuário") e outros detalhes opcionais como cardápio ("churrasco") seria capaz de concluir com algum grau de certeza que a foto pertence a esse evento de aniversário ("se a foto do usuário foi tirada em determinado momento, e nesse mesmo momento o usuário estava em um aniversário, então a foto do usuário foi provavelmente tirada nesse aniversário"). Adicionando essa nova informação na "rede de dados", uma aplicação de galeria de fotos poderia constatar que a foto retrata um churrasco ("se a foto foi tirada no dito aniversário, e no dito aniversário teve um churrasco, então existe a chance de que o churrasco esteja na foto").

A expansão dessa rede de dados por diferentes aplicações permite uma riqueza de informação e um grau de independência do usuário difícilmente alcançáveis pela forma como aplicações gerenciam seus dados atualmente, controlando-os e mantendo-os para si em repositórios de dados privados.

2.3.3 Resource Description Framework

Considerando a necessidade de atribuição de significado e de construção de relações entre entidades para a formação de uma "rede de dados", a W3C recomenda o Resource Description Framework (RDF) como modelo de dados para aplicações que precisam atribuir semântica ao seus dados.

O modelo RDF trabalha em torno do conceito de recursos, que são, basicamente, qualquer coisa identificável: uma pessoa, uma página Web, um átomo, eletricidade, gravidade, altura, um planeta, o universo, um deus, medo, a cor amarela e assim por diante. Esses recursos são identificados através de *Uniform Resource Identifiers* (URI), sequências de caracteres que identificam um recurso de forma uniforme através de um esquema (URI *scheme*). Ao se inserir um endereço de uma página Web em um navegador, como "http://example.com", esse endereço completo representa a URI (identificador) de uma página (recurso). O esquema "http" representa o protocolo HTTP, utilizado frequentemente para a obtenção de recursos HTTP, normalmente na forma de páginas Web. Especificar a sintaxe e a semântica de um esquema é de responsabilidade da entidade ou organização responsável pelo esquema. Em RDF, esquemas normalmente se referem a um determinado domínio, contendo definição de instâncias, classes, atributos e relações entre recursos.

URIs e esquemas são convenientes para serem usados na iniciativa da Web semântica porque delegam parte da necessidade de definir recursos para as entidades

que definem os esquemas. Isso permite que uma aplicação Web possa associar significado a um recurso sem precisar empregar grande esforço, uma vez que é possível apenas fazer referência a um recurso externo que representa o significado. Uma rede social, por exemplo, poderia associar o conceito do mundo real de "pessoa" ao usuário de nome "Goku" para ser exposto para outras aplicações. Isso pode ser feito simplesmente marcando um *namespace* "foaf" com referência à linguagem FOAF (*Friend of a Friend*) e código em notação RDF/XML. O trecho em notação RDF/XML resultante seria algo como:

```
<foaf:Person xmlns:foaf="http://xmlns.com/foaf/0.1/">
    <foaf:name>Goku</foaf:name>
</foaf:Person>
```

Caso uma aplicação precise descobrir o que é uma pessoa ou qual a relação entre uma pessoa e o seu nome, basta ela consultar o vocabulário referenciado pelo namespace "foaf". O uso de URIs e esquemas permite de um modo geral que uma aplicação precise especificar apenas informações de interesse sobre instâncias (ex. "o nome do usuário é Goku") e não propriedades genéricas de um conceito (ex. "pessoas são seres vivos").

Com a questão de mapeamento de entidades do mundo real em dados basicamente resolvida atráves de URIs e esquemas, ainda é necessário solucionar a questão de como esses dados se relacionam e formam uma "rede de dados". O RDF busca solucionar esse problema através da declaração de triplas no formato sujeito, predicado e objeto. Utilizando o exemplo do usuário de nome "Goku" citado anteriormente, suponhamos que ele seja o usuário da rede social de número 8001. O recurso que representa esse usuário tem como URI "dbz:user8001", com o esquema "dbz" definido e mantido pela aplicação de rede social. Para ilustrar a relação que o usuário número 8001 é uma pessoa, a aplicação pode definir a tripla "<dbz:user8001> <rdf:type> <foaf:Person>", representando respectivamente o sujeito, o predicado (a relação entre o sujeito e o objeto. Neste caso, o tipo do sujeito) e o objeto. Para a aplicação representar a relação que o nome do dito usuário é "Goku", o predicado "foaf:name" (que representa um nome no mundo real) seria usado, resultando na tripla "<dbz:user8001> <foaf:name> "Goku"" (com "Goku" considerado um literal, sem uma URI associada).

A partir de declarações de triplas RDF, relações entre recursos mais complexas do mundo real podem ser modeladas, formando uma verdadeira "rede de dados": "Goku (dbz:user8001) conhece (foaf:knows) Vegeta (dbz:user8000)", "dbz:user8000 é uma pessoa", "dbz:user8000 tem nome Vegeta", "Vegeta conhece Bulma", "Bulma conhece Chichi", "Chichi conhece Goku", etc. Triplas RDF podem ser representadas graficamente através de um grafo direcionado, com arestas (predicados) ligando nodos (sujeitos e objetos). A Figura 2.2 mostra uma representação gráfica simplificada do grafo resultante das triplas descritas neste parágrafo, com outras informações adicionais.

Conforme aplicações expõem dados no modelo RDF em uma "rede de dados", outras aplicações podem fazer uso dessas informações para enriquecer seus próprios serviços e até mesmo realizar tarefas automatizadas, uma vez que informações expostas são compreensíveis (portanto, usáveis) por máquinas, sem necessitar direção do usuário. No entanto, considerando o tamanho da Internet como um todo (e a quantidade de recursos e relações existentes), descobrir informação relevante para uma aplicação se torna difícil sem uma forma de consultar dados que trabalhe sobre dados no modelo

RDF, ou seja, realizar buscas por informação levando semântica em conta. Com isso em mente, a W3C recomenda como linguagem de consulta padrão o SPARQL Protocol and RDF Query Language, normalmente referido simplesmente como **SPARQL**.

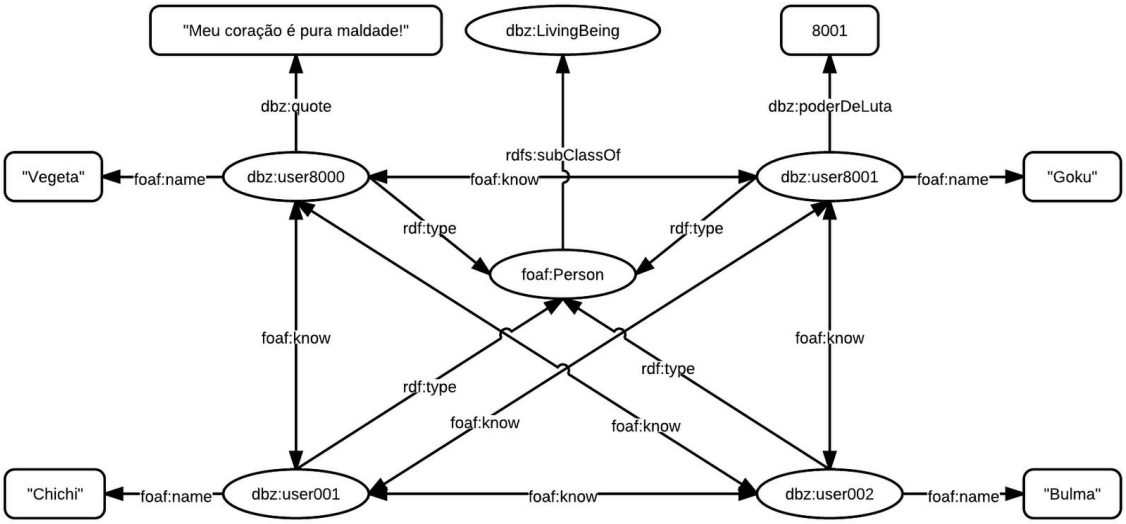

Figura 2.2: Representação de um modelo na forma de um grafo direcionado.

2.3.4 SPARQL Protocol and RDF Query Language (SPARQL)

O SPARQL é uma linguagem de consultas que trabalha sobre dados no modelo RDF e é uma das tecnologias que impulsionam o movimento da Web semântica. Diferente das linguagens de consulta normalmente utilizadas em bancos de dados relacionais (baseadas no SQL) que usam comparações de valores para filtrar registros, o SPARQL consulta um banco de dados com suporte ao modelo RDF através de padrões (apesar de um número considerável de implementações de bancos de dados com suporte ao RDF serem construídas sobre bancos de dados relacionais). Fazendo uma analogia com grafos, um banco de dados com suporte ao modelo RDF é representado como um conjunto de um ou mais grafos onde nodos são recursos (em termos de RDF, sujeitos e objetos de uma tripla) e arcos são relações entre esses recursos (predicados). Uma consulta SPARQL também pode ser considerada um grafo e registros que atendem os parâmetros de filtragem (um *match*) são, basicamente, subgrafos do banco de dados que são semanticamente equivalentes ao grafo de consulta.

O porquê da expressão “semanticamente equivalente” ser utilizada, e não “igual”, pode ser melhor compreendida com o seguinte exemplo: considerando o modelo representado como um grafo apresentado na figura 2.3, uma consulta SPARQL que “retorna os nomes de todos os seres humanos da espécie Homo sapiens” e outra que “retorna todos os nomes de seres humanos” retornariam resultados diferentes se o significado de cada entidade e relação não fosse levada em conta. A primeira consulta retornaria um resultado contendo o nome de uma instância de ser humano (“Derp”) enquanto que a segunda consulta retornaria um conjunto vazio porque, a nível sintático, não existe nenhuma entidade que possui um nome que é instância direta de um ser vivo. Ao se considerar semântica no modelo, as duas consultas retornariam o mesmo resultado (“Derp”) porque uma instância de ser humano é “semanticamente equivalente” a um ser vivo da espécie “Homo sapiens”.

Por ser capaz de considerar o significado das entidades existentes em um conjunto de dados durante o processamento de uma consulta, SPARQL é capaz de realizar consultas complexas, a par da complexidade das relações entre entidades do mundo real. Por esse motivo, a linguagem SPARQL é a linguagem padrão de consultas para dados no modelo RDF.

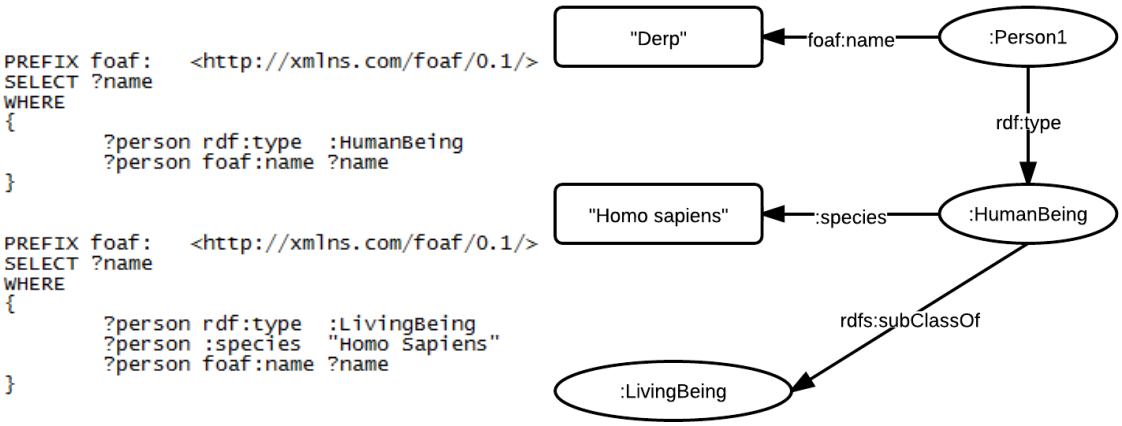

Figura 2.3: Ambas as consultas retornam o mesmo resultado em um modelo RDF.

2.3.4.1 SPARQL/Update (SPARUL)

SPARUL é uma extensão da linguagem SPARQL que adiciona as funcionalidades de inserção, deleção e atualização de triplas RDF em um modelo. A figura 2.4 apresenta um exemplo de representação em grafo de um modelo RDF no seu estado original. A figura 2.5 mostra o mesmo grafo após a operação de inserção da tripla “dbz:user7999 :status “vivo””. A figura 2.6 apresenta o resultado da execução de uma operação de atualização, composta por uma deleção seguida por uma inserção, resultando na substituição da tripla “dbz:user7999 :status “vivo”” para “dbz:user7999 :status “morto””. A figura 2.7 mostra o grafo resultante de uma operação de deleção que remove todas as triplas cujo sujeito é “dbz:user7999”.

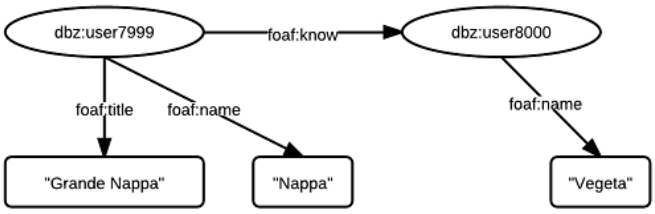

Figura 2.4: Modelo RDF representado como um grafo.

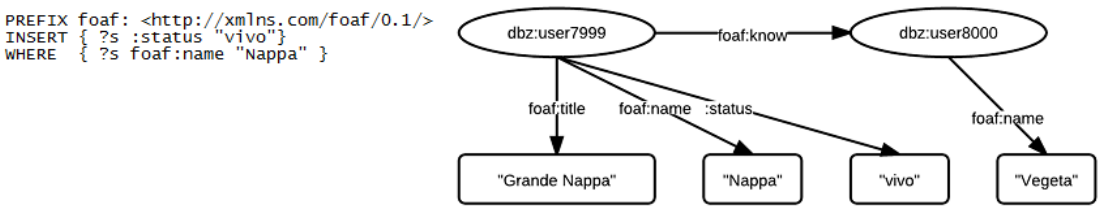

Figura 2.5: Inserção da relação “dbz:user7999 :status “vivo””.

```
PREFIX dbz:  <dbz:characters#>
DELETE {dbz:user7999 :status "vivo"}
INSERT {dbz:user7999 :status "morto"}
```

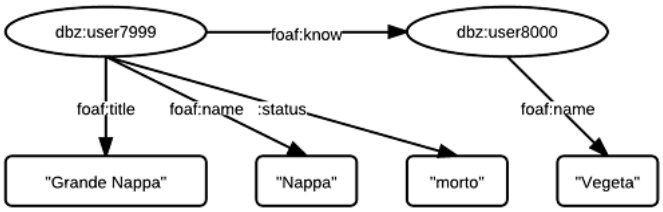

Figura 2.6: Atualização do “:status” de “vivo” para “morto”.

```
DELETE {?s ?p ?o}
WHERE
{
        ?s ?p ?o .
        ?s :status "morto"
}
```

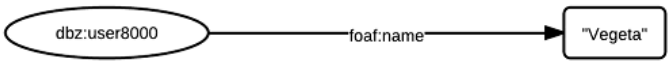

Figura 2.7: Deleção das declarações que envolvem recursos com “:status” “morto”.

2.4 Google Android

O Google Android é uma plataforma open-source para telefones celulares e outros dispositivos cujo objetivo é garantir a disponibilidade de uma plataforma aberta para que operadoras, desenvolvedores e a indústria possam tornar suas ideias realidade, sem depender das restrições ou do controle de uma única entidade (http://source.android.com/about/philosophy.html).

É importante ressaltar que é fora do escopo deste trabalho a abordagem minuciosa de como os mecanismos da plataforma funcionam ou de detalhar cada etapa do ciclo de vida de uma aplicação no sistema operacional, tendo em vista que tais detalhes são de pouca relevância para a proposta de implementação deste trabalho. Para a compreensão das próximas seções, o conhecimento de alguns termos da plataforma Android utilizados ocasionalmente neste trabalho são necessários:

- *Activity*: uma tarefa com uma interface gráfica que interage e captura informações vindas do usuário. Elementos visuais podem ser gerados dinamicamente ou através de um arquivo de layout que descreve uma interface gráfica
- *Intent*: mensagens assíncronas que são usadas para a comunicação entre aplicações. *Intents* podem ser enviados para o sistema operacional para executar aplicações, para requistar e receber informação de outra aplicação ou para sinalizar eventos
- *Service*: instância de uma classe desenvolvida para a realização de uma ou mais tarefas de longa duração, como um cronômetro que executa em plano de fundo enquanto o usuário interage com a interface gráfica
- *IntentService*: um *Service* que é executado somente quando solicitado através de um *Intent*. Após a realização da tarefa que lhe foi designada, o *IntentService* é liberado da memória. Por não estarem executando na maior parte do tempo, consomem menos memória, apesar da inicialização de um *IntentService* ser necessária toda vez que for instanciado

- *BroadcastReceiver*: instância de uma classe semelhante a um *Service* que captura *Intents* que possuem atributos definidos em um arquivo de *Manifest*. Ao receber um *Intent* do sistema operacional, realiza uma tarefa designada na definição da classe
- *HashMap*: um *array* associativo que mapeia pares de chave e valor
- *View*: objeto com representação gráfica instanciado em uma *Activity* com uma função específica. Exemplos de *Views* são *TextViews*, que exibem fragmentos de texto, *ImageViews*, que exibem imagens e *WebViews*, que carregam conteúdo de páginas Web em *Activities*
- *Layout*: definição em formato XML de uma interface gráfica. Normalmente utilizada para facilitar a criação de interfaces gráficas para Activities. Arquivos de *Layout* são normalmente compostos por *Views*
- *Manifes*t: arquivo que registra quais *Activities*, *Services*, *BroadcastReceivers* e outras classes estão associadas com uma aplicação, o comportamento dessas classes, e quais são os atributos que identificam os *Intents* que devem processar

O entendimento desses conceitos já são considerados suficientes para a compreensão deste trabalho. Informações que vêm a seguir são informações complementares dos conceitos listados anteriormente e da plataforma.

2.4.1 Intents

Uma característica bastante relevante para este trabalho é de que a plataforma dá forte ênfase para a comunicação e a cooperação entre aplicações através do uso de chamadas de intenção (*Intent*). *Intents* podem ser usados para realizar a comunicação entre aplicações através de mensagens e para delegar tarefas para outras aplicações, poupando os esforços repetidos necessários para implementar uma funcionalidade que já existe em outras aplicações.

Intents são descrições de operações a serem executadas que podem ser enviadas por aplicações de forma explícita ou implícita para o sistema operacional. *Intents* explícitos são intenções de execução de uma tarefa endereçada a uma classe ou aplicação específica. Um exemplo de *Intent* explícito é quando uma aplicação precisa invocar um *Service* específico para realizar uma tarefa em plano de fundo, como capturar a próxima mensagem SMS que o dispositivo receber ou iniciar a contagem de um cronômetro interno.

Intents implícitos são intenções de realização de uma determinada tarefa por qualquer aplicação que possa realizar essa tarefa. Assumindo que o dispositivo de um usuário possui múltiplos clientes de e-mail, a abertura de um *link* em um navegador com o esquema URI “mailto” (por exemplo, “mailto:someone@example.com”) provocaria o envio de um *Intent* implícito com a intenção de “enviar um e-mail” para o sistema operacional. A partir das informações declaradas nos arquivos *Manifest* das aplicações instaladas, o sistema operacional elabora uma lista das aplicações que conseguem realizar a tarefa descrita no *Intent* (tarefa declarada nos campos *action* e *categories*) e apresenta essa lista ao usuário. O usuário então escolhe qual a aplicação de sua preferência para executar a tarefa, executando a aplicação selecionada usando os argumentos enviados pelo *Intent* (campo *extras*). Um exemplo de lista de aplicações disponíveis para envio de e-mails é apresentada na figura 2.8.

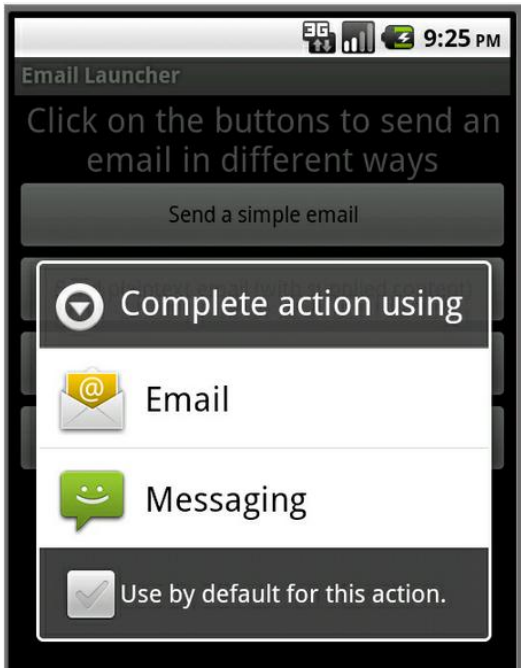

Figura 2.8: Exemplo de lista para a seleção de uma aplicação para enviar e-mails.

2.4.2 Sandboxing

Para evitar acesso indevido da informação por outras aplicações, o Android oferece isolamento de processos através do uso de *sandboxing*, um mecanismo de segurança que permite a execução de código não-confiável minimizando os recursos disponíveis para uso dessa aplicação: qualquer funcionalidade não-padrão deve ser explicitamente requisitada pela aplicação, como acesso à Internet, envio de SMS em nome do usuário e ler a lista de contatos.

Devido à forma como aplicações são isoladas dentro do sistema, as estratégias de armazenamento de dados de uma aplicação não possuem funcionalidades que permitem que aplicações compartilhem dados em um único repositório, da mesma forma que aplicações não podem acessar os repositórios de dados privados de outras aplicações. Quando aplicações precisam compartilhar dados, normalmente são usados *Intents* ou *ContentProviders*, que expõem uma interface que implementa as operações CRUD. É de responsabilidade do criador do *ContentProvider* de implementar essas operações.

Das quatro formas básicas de armazenamento local de dados, apenas o uso de arquivos externos permite um certo grau de compartilhamento de dados sem o uso de um protocolo de comunicação explícito. As outras formas, *shared preferences* (persistência de pares de chave-valor), arquivos internos (área restrita do sistema de arquivos) e banco de dados SQLite (banco de dados relacional) são restritos à aplicação que criou esses dados.

3 ARQUITETURA

A arquitetura descrita neste capítulo é originalmente detalhada por Brodt et al. (2011). Os autores do trabalho propõem uma arquitetura para gerência de dados para dispositivos móveis que busca oferecer interoperabilidade a nível de dados tanto para aplicações internas, como aplicações nativas e sensores do dispositivo, quanto para aplicações externas, como páginas Web (acessadas pelo navegador) e aplicações de outros dispositivos vizinhos.

A arquitetura é basicamente uma expansão do Telar DCCI (BRODT e NICKLAS, 2008), descrito brevemente na seção 1.4 deste trabalho, com a adição de mecanismos que permitem que aplicações em ambientes diferentes sejam interoperáveis a nível de dados e não apenas que páginas Web possam receber leituras do sensor de GPS do dispositivo. Outras funcionalidades incluídas na arquitetura são a indexação eficiente de dados dinâmicos, um forte suporte para a realização de consultas espaciais (*queries* onde a localização geográfica de um registro ou de uma entidade são parâmetros de filtragem), mecanismos de controle de acesso aos dados e uma implementação de um banco de dados com suporte ao modelo de dados RDF e à linguagem de consultas SPARQL. Uma visão geral da aruitetura pode ser vista na figura 3.1.

Este capítulo dará ênfase nos aspectos da arquitetura que são pertinentes à implementação proposta no capítulo 4. Primeiramente, é dada uma visão geral da organização da arquitetura, mais especificamente, sobre o repositório de dados unificado, responsável por possibilitar a interoperabilidade entre aplicações diversas e também o componente principal da arquitetura. Em seguida, são detalhados os mecanismos que possibilitam que as aplicações consigam interagir com o repositório e realizar operações sobre ele.

3.1 Integrated Resource and Context Repository

O repositório unificado, originalmente denominado "*Integrated Resource and Context Repository*", é uma implementação de banco de dados com suporte a dados no modelo RDF que tem como objetivo atuar como a base de dados central do dispositivo. Esse repositório não implica necessariamente em eliminar o uso de bancos de dados privados em aplicações, uma vez que determinadas aplicações precisam manter informações para si: uma rede social, por exemplo, perderia o sentido se todos os dados do perfil de um usuário estivessem apenas em seu dispositivo. Qualquer visualização do perfil por outro usuário exigiria uma consulta dentro do repositório unificado interno ao dispositivo do dono do perfil. Caso o aparelho esteja por algum motivo, indisponível, a visualização se tornaria impossível.

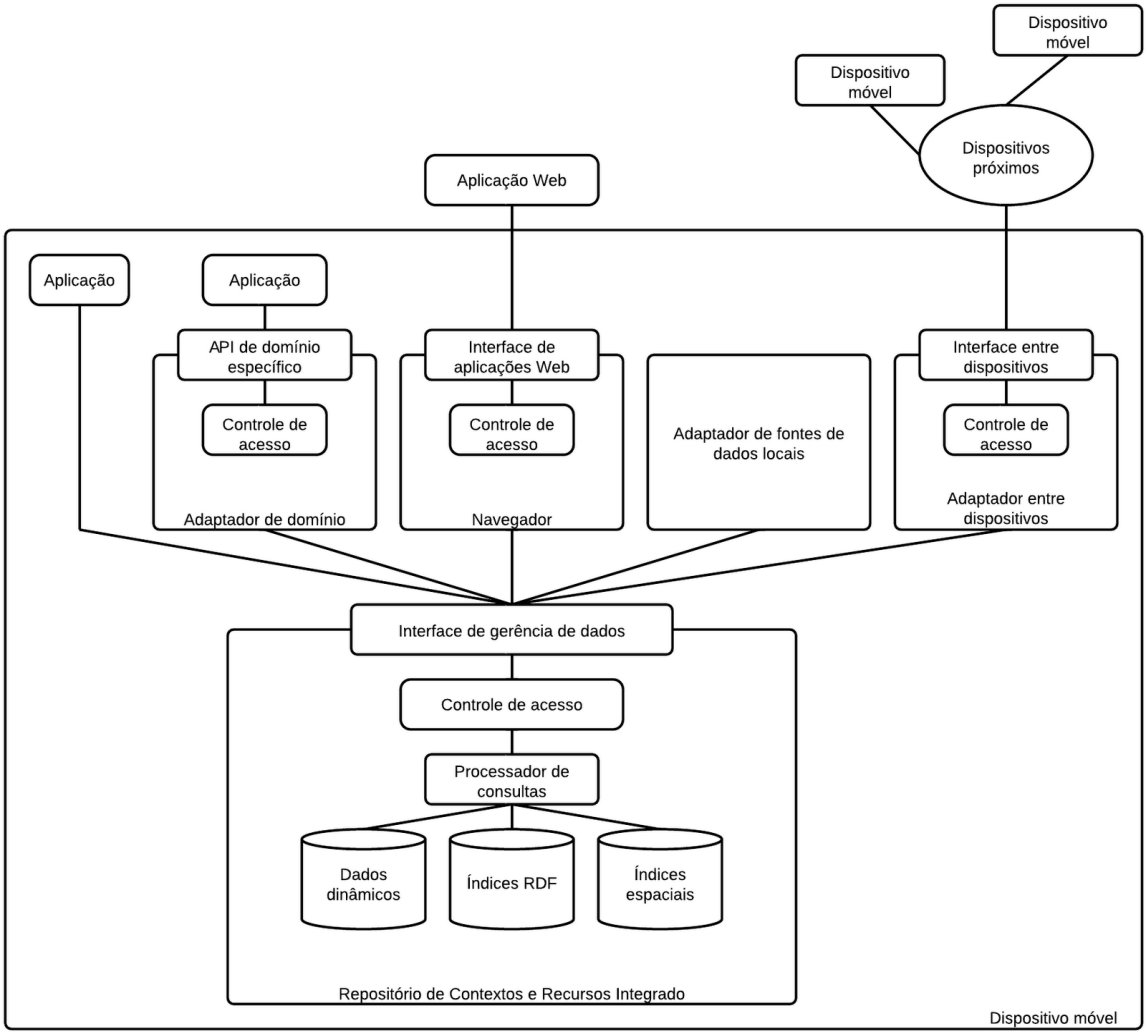

Figura 3.1: Visão geral da arquitetura proposta por Brodt et al., 2011.

Logicamente, a criação de múltiplas cópias de dados abre espaço para o surgimento de redundância e, possivelmente, inconsistência, mas o repositório unificado possui a capacidade de gerenciar esse tipo de redundância graças ao modelo de dados RDF. O uso do modelo de dados RDF permite que informação seja exposta para outras aplicações em linguagem legível e compreensível por máquinas, facilitando enormemente a interação automatizada entre aplicações.

Ainda considerando o exemplo do perfil de um usuário em uma rede social, existiria uma cópia local dos dado no repositório e uma outra cópia em um servidor remoto. Quando o usuário se conecta à página Web dessa rede social e acessa o próprio perfil, a página pode solicitar uma consulta ao repositório sobre dados que são pertinentes no contexto de redes sociais como informações pessoais do usuário. Caso o usuário tenha concedido permissão para a página consultar o repositório, o navegador (que faz o intermédio entre a página e o repositório) consulta a base de dados e retorna o resultado para a página Web. Considerando que os dados no modelo RDF recebidos são compreensíveis pela aplicação, a página Web pode automaticamente atualizar as informações do perfil no seu banco de dados privado e finalmente apresentar essa informação ao usuário. Ao final desse processo, as informações estarão em um estado consistente, apesar de serem redundantes, e praticamente sem esforço do usuário.

Em suma, é desejável que apenas informação destinada a ser compartilhada seja adicionada ao repositório unificado, embora o seu uso como um banco de dados interno de uma aplicação ser plenamente possível e que o repositório unificado pode ser usado também apenas para consulta, com um repositório de dados interno separado.

3.1.1 Operações e Controle de Acesso

O processador de consultas do repositório suporta a linguagem de consultas SPARQL com extensão para SPARUL, apresentados no capítulo 2. Como originalmente definido, o repositório unificado possui também suporte ao armazenamento de índices espaciais e atualização eficiente de dados dinâmicos, como leituras de GPS. Tais funcionalidades não são implementadas na parte da implementação deste trabalho e, portanto, não serão detalhadas.

Considerando que na arquitetura proposta, praticamente todos os dados a serem compartilhados estão no repositório unificado, considerações relacionadas à segurança surgem. Primeiramente, o repositório unificado deve possuir capacidades de determinar quais são os privilégios que uma aplicação possui antes de realizar operações, de forma que operações sejam consideradas inconsistentes quando trabalham sobre dados dos quais a aplicação não possui privilégios de leitura ou escrita. Autenticação de aplicações Web é feita através de certificados SSL com protocolo de comunicação via HTTPS, enquanto que aplicações nativas do dispositivo são autenticadas através de um identificador único para cada aplicação e uma assinatura digital do fabricante. Dispositivos próximos são identificados a partir do código do dispositivo.

Além disso, é necessário que aplicações sejam autenticadas antes de operarem sobre o repositório, de forma a evitar que uma aplicação se passe por outra para acessar dados dos quais não possui previlégios de leitura ou de escrever dados em nome de outra aplicação. É necessário também que o usuário tenha a possibilidade de gerenciar permissões (preferencialmente através de um interface gráfica intuitiva), de forma a revogar permissões anteriormente concedidas a determinadas aplicações ou de conceder priviégios a outras conforme a sua vontade.

Apesar da implementação proposta neste trabalho não abordar tais estratégias de segurança, elas são descritas brevemente neste capítulo pela sua importância para trabalhos futuros e para fins informativos. Maiores detalhes são encontrados no trabalho original (BRODT et al., 2011).

Como padrão, aplicações possuem privilégios de escrita e leitura sobre dados que elas mesmas criam e qualquer outra informação ou recurso a serem acessados devem ser explicitamente solicitados ao usuário, exibindo todas as permissões das quais necessita. A arquitetura implementa um mecanismo de permissões a partir do princípio de controle de acesso baseado em *access roles* proposta por FERRAIOLO e KUHN (1992). Quando o usuário concede permissões a uma aplicação para trabalhar sobre dados, essa aplicação recebe diversos “papéis” (no mesmo sentido de um ator que desempenha diversos papéis em uma peça de teatro) que contêm a definição das permissões propriamente dita.

A figura 3.2 apresenta graficamente o conceito de *access roles* como recursos no modelo de dados RDF. AccessRole30 concede permissão de escrita em instâncias da classe “foaf:Person” para “http://example.com”, da mesma forma que AccessRole56 concede permissões de leitura da propriedade “foaf:name” em qualquer instância de

“foaf:Person”. ClassPropertyList1 agrupa classes e propriedades que caracterizam o objeto alvo da AccessRole56.

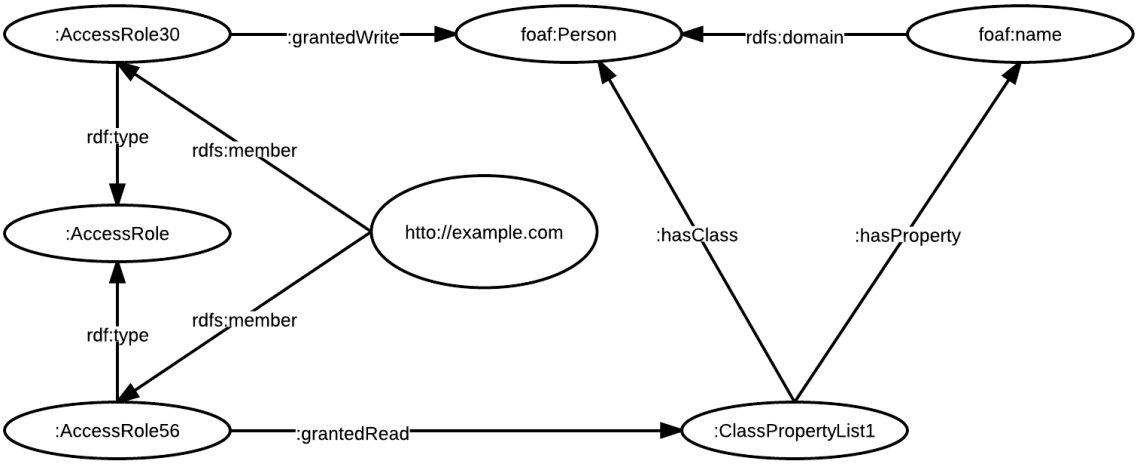

Figura 3.2: Exemplo de instâncias de *access roles*.

Considerando que o modelo RDF se baseia na descrição de relações entre dados, é consideravelmente simples definir e trabalhar sobre metadados, especificando controle de acesso a uma determinada classe de instâncias ou a um conjunto de propriedades. definidos em uma lista de classes e propriedades.

3.1.2 Interface de Gerência de Dados

O repositório de dados unificado recebe todas as requisições de operações através de uma interface exposta diretamente para as aplicações nativas do dispositivo ou para os diversos adaptadores existentes (vistos na próxima seção). Os principais métodos disponibilizados por essa interface são as funções de consulta (SPARQL) e atualização (SPARUL), tendo também outras funções de suporte como funções para registros de *listeners*, onde aplicações são notificadas sobre a modificação de dados dinâmicos, e métodos para atualização de dados obtidos do adaptador de fontes de dados locais sem a necessidade do processamento de operações em SPARUL (atualização de nível binário).

3.2 Provisão de Dados

3.2.1 Adaptadores

A arquitetura utiliza o conceito de adaptadores que realizam o intermédio entre as aplicações e o repositório de dados. O objetivo principal dos adaptadores é de prover uma interface com abstrações de alto nível para as aplicações manipularem o repositório de dados unificado. Um exemplo de adaptador para páginas Web são os navegadores. Na arquitetura proposta, são definidos quatro tipos de adaptadores, que possuem funções variadas. São eles:

- Adaptador de domínio: encaminha operações de manipulação do repositório e abstrai a criação de operações de manipulação do repositório, evitando que uma aplicação nativa do dispositivo envie uma consulta gerada dentro da aplicação. Como as operações foram criadas pelo adaptador, e não pela aplicação, a

operação pode ser considerada segura de ser executada, evitando assim a necessidade de controle de acesso e consequente modificação da operação para se adaptar às permissões de acesso

- Navegador: além das funções convencionais de um navegador, autentica as páginas que desejam encaminhar operações ao repositório de dados e encaminha apenas as operações de aplicações que possuem privilégios para consultar o repositório
- Adaptador de fontes de dados locais: encaminha informação obtida dos diversos sensores existentes no dispositivo para serem atualizados no repositório, na área destina a dados dinâmicos (cuja indexação é ineficiente por causa do alto número de atualizações). Envia atualizações a nível binário para otimizar o tempo de atualização de dados dinâmicos, uma vez que o processamento de operações em SPARQL/SPARUL é consideralvemente mais lento
- Adaptador entre dispositivos: controla acesso de requisições originadas de outros dispositivos e as encaminha para o repositório.

Apesar de adaptadores de domínios serem usados por aplicações nativas do dispositivo, o uso dele não é obrigatório. Aplicações nativas podem encaminhar operações diretamente ao dispositivo utilizando os métodos expostos pela interface de gerência de dados.

3.3 Impacto da arquitetura sobre aplicações

Aplicações nativas do dispositivo e aplicações Web que realizam operações sobre o repositório precisam ser implementadas com interoperabilidade em mente.

Aplicações Web precisam de funcionalidades que permitem criar consultas na linguagem SPARQL e operações em SPARUL para serem encaminhadas pelo navegador. O navegador recebe o resultado retornado de uma operação e a processa, encapsulando-a em um objeto de uma linguagem de programação compreensível por páginas Web como Javascript, o que evita que a página precise empregar esforços para o processamento do conjunto de resultados, antes de usá-los.

Aplicações nativas do dispositivo precisam gerar as próprias operações em SPARQL/SPARUL, caso não utilizem adaptadores de domínios. Ao receber os resultados da operação, essas aplicações precisam processar o conjunto de resultados em XML ou transformá-lo em um objeto manipulável, como um iterador. O uso de um adaptador permite que aplicações recebam como resultado um objeto pré-processado mais simples de ser utilizado pela aplicação

4 PROPOSTA E IMPLEMENTAÇÃO

Este capítulo apresenta em detalhes a implementação de um protótipo de sistema com base na arquitetura proposta por Brodt et al., apresentada no capítulo 3. Primeiramente, é descrito o *SparqlForwarder*, implementação desenvolvida durante intercâmbio na universidade de Stuttgart e precursor da implementação realizada neste trabalho. Depois, são descritos os detalhes da implementação do repositório de dados unificado, do protótipo de navegador (que implementa parcialmente as funcionalidades do *SparqlForwarder*) e dos protótipos de aplicação.

4.1 SparqlForwarder

A proposta de implementação descrita neste capítulo é fundamentalmente uma variação da implementação de um projeto de pesquisa como parte da tese de doutorado de Andreas Brodt, desenvolvido na Universidade de Stuttgart (Universität Stuttgart), durante o acordo de intercâmbio entre o Instituto de Informática da UFRGS e o Instituto de Sistemas Paralelos e Distribuídos da Uni-Stuttgart, sob supervisão do professor Bernhard Mitschang, no período entre fevereiro de 2011 até janeiro de 2012.

A implementação desenvolvida durante o projeto, originalmente chamada de "SparqlForwarder" (posteriormente renomeada para simplesmente "Repository"), consiste em uma extensão para o navegador da Fundação Mozilla, o Mozilla Firefox (para computadores pessoais) com o objetivo de tornar o navegador um protótipo de repositório de dados unificado para páginas Web, utilizando o modelo de dados RDF como modelo de dados padrão.

O *SparqlForwarder* foi desenvolvido pelo autor deste trabalho juntamente com o aluno Bruno Romeu Nunes, ambos alunos do Instituto de Informática da UFRGS. Foi utilizada uma metodologia de desenvolvimento incremental, baseada na execução de um conjunto de pequenas tarefas de implementação, estas sendo apresentadas ao coordenador do projeto e então integradas ao restante do sistema.

O Mozilla Firefox foi escolhido como plataforma da implementação devido ao modelo de componentes do *framework* da Fundação Mozilla, o *Cross Platform Component Object Model*, XPCOM. Esse modelo de componentes oferece um alto nível de modularidade para a integração de novas funcionalidades. Por isso, é utilizado frequentemente para o desenvolvimento de extensões para os produtos da Fundação Mozilla. Existem diversas referências e tutoriais disponíveis na Internet, de forma que a implementação possa ser realizada de forma bastante direta e objetiva, mesmo considerando a complexidade do modelo em si.

Para a implementação das funcionalidades do repositório unificado, foi utilizada a biblioteca RDFLib, desenvolvida em Python, que oferece classes e métodos para a

manipulação de modelos RDF, além de persistência de dados, o que permite que alterações nos dados do repositório possam ser propagadas para outras aplicações. Para se utilizar a biblioteca RDFLib de forma nativa, foi necessário a extender o navegador com a extensão Pythonext, que realiza o *binding* da linguagem Python no navegador, de forma que Python possa ser processado dentro de um componente implementado em XPCOM.

4.1.1 Funcionamento

A função do *SparqlForwarder* é de expor uma interface para aplicações Web (a "*Web Application Interface*" da arquitetura descrita no capítulo 3) no navegador de forma a intermediar a comunicação entre uma página Web e o repositório de dados unificado (interno ao navegador). Por isso o "*Forwarder*" ("encaminhador") no nome da extensão. Como a interação com os dados no repositório é feita através de operações de consulta e atualização em SPARQL e SPARUL, a aplicação implementada pode ser definida como "aquele que encaminha (consultas) SPARQL".

Uma aplicação Web inicialmente solicita um objeto ("*repositoryHandle*") para realizar operações no repositório ao navegador através da interface exposta pela extensão (em uma variável global "*repository*"). Ao solicitar a abertura de uma conexão com o repositório de dados, a página envia também as *AccessRoles* (vide capítulo 3) das quais afirma precisar. A extensão verifica o certificado SSL da página Web de forma a autenticá-la e em seguida retorna à página as *AccessRoles* que a página tem de fato, junto com um *repositoryHandle*. No caso de essa ser a primeira vez que a página tenta utilizar o repositório, é exibida ao usuário uma caixa de diálogo com as *AccessRoles* requisitadas. O usuário tem então liberdade de determinar quais *AccessRoles* a página terá direito, se houver alguma. Com as *AccessRoles* retornadas pelo *SparqlForwarder*, a aplicação é capaz de determinar quais operações ela pode realizar (operações que trabalham sobre dados das quais a aplicação não possui direito são consideradas inconsistentes e, portanto, não processadas pelo repositório).

Caso a aplicação deseje escrever dados no repositório, ela chama o método "*executeUpdate*" do *repositoryHandle* tendo como argumento uma operação em SPARQL/Update na forma de uma *string*. O processador de operações do repositório executa a operação de escrita, caso a aplicação possua os privilégios necessários.

Caso a aplicação deseje realizar uma consulta sobre o repositório, ela realiza a chamada do método "*executeQuery*" do *repositoryHandle* com uma *string* contendo uma consulta na linguagem SPARQL, recebendo como resultado uma lista de resultados onde cada elemento possui os dados requisitados na cláusula SELECT que estão de acordo com os parâmetros de filtragem. Uma vez estruturados em um formato usado pela aplicação, os dados estão prontos para serem usados.

4.2 Proposta de Implementação

Considerando que o repositório unificado interno ao navegador é o mesmo para todas as aplicações Web que operam sobre dados compartilhados, dados criados por uma aplicação são, em teoria, visíveis para todas as outras (na prática, visíveis apenas por aplicações que possuem as devidas permissões). Como os dados são modelados no formato RDF e esses dados são compreensíveis por máquinas, o repositório oferece interoperabilidade a nível de dados entre aplicações.

No entanto, *SparqlForwarder* foi desenvolvido para oferecer interoperabilidade a nível de dados apenas para aplicações Web. Da forma como foi implementado, aplicações nativas de um dispositivo móvel são incapazes de operarem sobre dados existentes, tampouco adicionar novos dados. Extender o *SparqlForwarder* para suportar chamadas de operações de aplicações nativas da plataforma de um dispositivo móvel exigiria modificações tanto na extensão como no próprio navegador. Como a extensão foi desenvolvida para ser utilizada apenas no Mozilla Firefox 5.0, e esse navegador é atualmente considerado ultrapassado, o desenvolvimento de uma nova aplicação aparenta ser uma alternativa mais interessante.

Levando esses fatores em conta, é possível considerar três possibilidades: implementar o novo repositório de dados unificado como uma aplicação nativa do dispositivo, extender o navegador com um repositório de dados ou, novamente, desenvolver um repositório como uma extensão para um navegador. Pela experiência obtida anteriormente com o desenvolvimento do *SparqlForwarder*, foi decidido realizar a implementação como uma aplicação nativa pelos seguintes fatores (entre outros):

- A Fundação Mozilla possui um ciclo acelerado de desenvolvimento de novas versões para o Mozilla Firefox (aproximadamente seis semanas entre o lançamento de duas versões). O *SparqlForwarder* foi desenvolvido para ser suportado por uma versão fixa do Mozilla Firefox (5.0) porque a extensão se tornava incompatível com o navegador de seis em seis semanas
- O modelo de componentes da Fundação Mozilla (XPCOM), apesar de modular, é consideravelmente complicado de usar pela quantidade de definições de interfaces em linguagem de baixo nível necessárias para que um componente consiga acessar funcionalidades de outro componente
- Usuários trocam de navegador de forma mais frequente que sistemas operacionais em um dispositivo móvel
- Um repositório de dados e um navegador possuem funções bastante diferentes. Portanto, não há razão para mantê-las juntas em uma única aplicação

Adicionalmente, o autor deste trabalho desenvolveu um interesse crescente pela plataforma Android, da Google. Com o objetivo secundário de adquirir maior conhecimento e experiência com a plataforma, a proposta definitiva de implementação deste trabalho pode ser definida conforme o seguinte:

"Desenvolver um repositório de dados unificado para dispositivos móveis capaz de fornecer interoperabilidade a nível de dados entre aplicações Web e aplicações nativas, com o objetivo de reduzir os problemas causados pela redundância e pela inconsistência de dados"

4.2.1 Diferenças entre Arquitetura e Implementação

Considerando a dimensão da arquitetura proposta anteriormente (capítulo 3) e a diversidade de áreas diferentes abordadas, a proposta de implementação foi consideravelmente reduzida para um nível condizente com o propósito deste trabalho como um trabalho de graduação. Inicialmente, era esperado que o *SparqlForwarder* servisse como uma implementação que validasse (parcialmente) a arquitetura proposta, sendo suficiente para a finalidade deste trabalho. Levando em conta os fatores que apontam para uma nova implementação mencionados anteriormente, a implementação foi reescrita para a plataforma Android na linguagem Dalvik, variante do Java, em

contraste com a extensão para o navegador Mozilla Firefox, desenvolvido com *bindings* de Javascript e Python para a tecnologia XPCOM.

Antes de nos aprofundarmos nos detalhes de implementação, é importante frisar que a implementação é um subconjunto das funcionalidades propostas na arquitetura apresentada no capítulo 3. A figura 4.1 apresenta graficamente as partes da arquitetura abordadas na implementação.

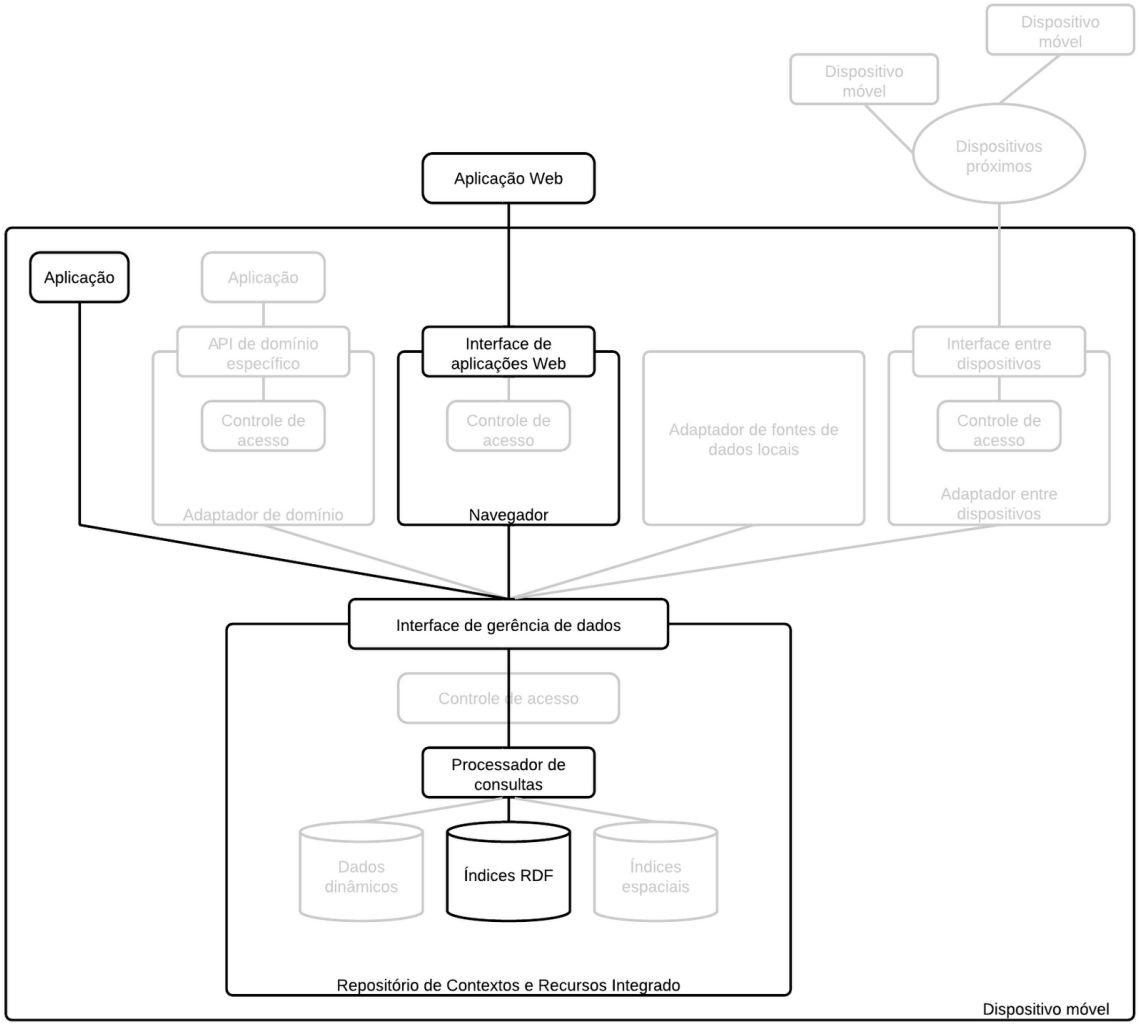

Figura 4.1: Partes implementadas da arquitetura proposta por Brodt et al.

A implementação foca na resolução de problemas de redundância e inconsistência de dados envolvendo aplicações Web e aplicações para dispositivos móveis, desconsiderando interoperabilidade com dispositivos externos e com sensores. Uma aplicação nativa desenvolvida para a plataforma pode, na maior parte das situações, simplesmente substituir os adaptadores entre dispositivos e de fontes de dados locais, intermediando a comunicação entre os sensores ou os dispositivos próximos com o repositório de dados.

Adicionalmente, adaptadores de domínios não foram implementados, uma vez que esses componentes são opcionais para se obter comunicação entre aplicações nativas e o repositório. A implementação não considera também o armazenamento diferenciado de

dados dinâmicos nem a criação de índices espaciais devido a pouca influência que elas possuem para alcançar interoperabilidade.

Considerando que a parte da implementação deste trabalho é basicamente a extensão do *SparqlForwarder* para a plataforma Android, e que a parte de controle de acesso e segurança da extensão foi feita pelo aluno Bruno Romeu Nunes, foi decidido não trabalhar a área de controle de acesso no navegador (por ter sido implementado por outra pessoa) nem no repositório, uma vez que o controle de acesso baseado em *AccessRoles* exige modificações das funcionalidades de consulta e atualização para que o resultado retornado a uma aplicação seja uma interseção do resultado original da operação com os recursos dos quais a aplicação possui permissão de operar. Tais modificações nos códigos do processador de operações SPARQL/SPARUL seriam suficientemente complexas para a elaboração de um trabalho à parte.

4.2.2 Funcionalidades Implementadas

- Repositório de dados unificado: composto de um banco de dados com suporte a modelos RDF e a operações nas linguagens SPARQL e SPARUL. O resultado retornado à aplicação (no caso de consultas) é um objeto contendo registros, cada registro contendo os valores obtidos das triplas RDF
- Interface de gerência de dados: responsável pela exposição do repositório às aplicações e ponto de entradas das operações no repositório
- Navegador: um protótipo de navegador desenvolvido como substituto do *SparqlForwarder*, disponibilizando uma interface (interface de aplicações Web) no qual aplicações Web podem requisitar chamadas de operações através do navegador. Dispensa o uso de uma extensão para adicionar a funcionalidade de comunicação entre o navegador e o repositório
- Aplicação nativa: um exemplo de aplicação nativa que realiza operações de inserção no repositório
- Aplicação Web: um exemplo de aplicação Web que realiza consultas no repositório.

4.3 Detalhes de implementação

A implementação foi desenvolvida a partir de tecnologias de áreas bastante diferentes. A linguagem Dalvik foi utilizada para o desenvolvimento da aplicação nativa de exemplo, do repositório de dados unificado e do navegador. A aplicação Web foi desenvolvida em HTML e possui lógica desenvolvida em Javascript. A interface que liga a aplicação Web e o navegador foi desenvolvida em Dalvik, mas quando utilizada por uma aplicação Web, a interface retorna dados traduzidos no formato JSON apropriados pra serem processados utilizando a linguagem Javascript.

4.3.1 Repositório

Em termos da plataforma Android, o repositório de dados unificado é mais uma aplicação nativa instalada no sistema. O repositório é basicamente um *IntentService* que captura *Intents* contendo operações de manipulação da base de dados, processa essas operações e envia os resultados à aplicação requisitante.

A interface de gerência de dados, existente na arquitetura detalhada no capítulo 3, é implementada implicitamente através do recebimento e envio desses *Intents*. Aplicações Web e aplicações nativas produzem uma operação em SPARQL/SPARUL e as enviam para o repositório contendo a ação "*tcc.action.Repository*", uma *string* contendo uma consulta SPARQL ou uma atualização SPARUL e uma *string* representando a ação que deve ser associada ao Intent enviado como resultado.

O *Intent* de resultado de uma consulta contém internamente um conjunto de resultados representado como uma lista de *HashMaps*, cada *HashMap* contendo vários pares de chave-valor com as variáveis associadas à cláusula SELECT como chaves e os valores obtidos de um *match* como os valores do par. Ao ser enviado, esse *Intent* de resposta tem o campo ação preenchido com a ação recebida da aplicação requisitante, de forma a permitir que a aplicação requisitante possa receber devidamente o resultado.

4.3.1.1 Operações e Modelo de Dados

Internamente, o repositório utiliza uma adaptação (*porting*) do *framework* Jena, criado originalmente pela Hewlett-Packard, para a plataforma Android chamada Androjena. Jena é um *framework* para desenvolvimento de aplicações que trabalham com a iniciativa da Web semântica em mente.

O Androjena provêm as funcionalidades para a manipulação de modelos RDF, enquanto que a capacidade de persistência em um banco de dados e o processamento de consultas e atualizações nas linguagem SPARQL e SPARUL sobre um modelo são realizadas através de componentes extras chamados TDBoid e ARQoid (*portings* dos componentes TDB e ARQ originais para o *framework* Jena).

Androjena ainda é um projeto em fase experimental, em contraste com o Jena que é mantido pela fundação Apache, e apresenta resultados inconsistentes dependendo da operação realizada. Testes com o Androjena apontam que a operação de deleção de triplas RDF de um dado modelo não são efetuadas dependendo dos argumentos na cláusula WHERE.

Adicionalmente, atualizações, que são remoções de triplas RDF seguidas de inserções de triplas modificadas, também apresentam comportamento anormal devido à dependência da operação de deleção. Outra inconsistência verificada é de que mudanças no modelo que sejam diferentes de inserções não são persistidas no repositório.

Dados esses problemas, a implementação deste trabalho aborda apenas as operações de inserção (feita por uma aplicação Android) e consulta (realizada por uma aplicação Web) nas aplicações de exemplo.

Ao receber uma operação, o repositório carrega o conjunto de dados RDF do banco de dados interno ao repositório. Um conjunto de dados pode conter múltiplos modelos RDF associados. Ao se escolher um modelo (nesta implementação, sempre o mesmo), ele é carregado em memória e está pronto para ser consultado ou modificado.

O Androjena oferece suporte para transações atômicas, mas essa funcionalidade não foi utilizada. A operação recebida através de um *Intent* de requisição é então processada sobre o modelo, obtendo-se um conjunto de resultados que pode ser percorrido através de um iterador, transformado em um outro modelo (para a realização de operações adicionais) ou convertido para o formato XML.

A partir do iterador, os resultados são serializados e adicionados ao Intent de resposta, sendo em seguida despachado para o sistema operacional para ser capturando pela aplicação requisitante.

4.3.2 Navegador

O navegador é uma aplicação nativa da plataforma composto por uma *Activity*, usada para exibir conteúdo para o usuário, um *BroadcastReceiver*, que monitora a chegada de *Intents* contendo resultados de operações realizadas pelo navegador em nome das aplicações e uma interface exposta às páginas Web que recebe requisições de operações e retorna dados traduzidos de Dalvik para Javascript.

O *framework* de desenvolvimento de aplicações para Android oferece um elemento de *layout* chamado *Webview* capaz de carregar o conteúdo de páginas HTML diretamente dentro de uma *Activity*. Apesar de não ser um navegador completo, o *Webview* possui suporte à exibição de páginas Web e de executar código em *Javascript*, necessário para que o navegador possa expor a interface com a qual páginas Web podem realizar requisições sobre o repositório de dados.

A parte da *Activity* do navegador contém um elemento *Webview* que carrega a aplicação Web e expõe uma variável global "*android*", por onde a aplicação pode invocar os métodos "*executeQuery*" para a realização de consultas, tendo como parâmetro uma *string* contendo uma consulta SPARQL, e "*executeUpdate*", contendo uma *string* com uma operação SPARUL para atualização de dados no repositório.

Apesar do envio de *Intents* ser uma funcionalidade assíncrona, a interface de aplicações Web expõe os métodos a serem invocados de forma síncrona. A chamada de uma operação de consulta ou atualização na interface encaminha de forma assíncrona um *Intent* para o sistema operacional para que chegue no repositório de dados. O navegador então mantém um *BroadcastReceiver* ativo que monitora a chegada do *Intent* de resultado do repositório. Uma vez obtido o resultado, a interface converte os dados para o formato JSON e encaminha o resultado para a aplicação.

Antes do envio do *Intent* contendo a operação, o navegador deveria realizar controle de acesso da aplicação, verificando seu certificado SSL. Com a aplicação autenticada, o *Intent* pode ser enviado para o sistema operacional. Como mencionado anteriormente, o controle de acesso não foi implementado neste protótipo. A implementação também não considera a situação de múltiplas aplicações Web realizando requisições em paralelo.

4.3.3 Aplicações Desenvolvidas

As aplicações desenvolvidas para este trabalho são aplicações em nível de protótipo, que realizam operações simples de consulta e inserção de dados mas que, em termos práticos, não refletem o comportamento de aplicações em cenários de relações complexas, característicos do mundo real, onde os conceitos de Web semântica podem ser aplicados com maior significância.

Foram implementados uma aplicação nativa da plataforma Android, que realiza operações de inserção de triplas RDF enviando *Intents* com operações de inserção em SPARUL, e uma página em HTML que acessa as informações do repositório utilizando a interface de aplicações Web exposta pelo protótipo de navegador. A página realiza uma chamada de consulta da interface, enviando como parâmetro uma consulta em SPARQL. Ao receber os dados obtidos da consulta, a aplicação apresenta uma janela *pop-up* com uma lista de pares de chave-valor para cada registro obtido do resultado.

Apesar dos resultados obtidos demonstrarem interoperabilidade real a nível de dados, um caso de teste bem desenvolvido torna-se necessário para validar devidamente esses resultados. Adicionalmente, é necessária a correção do comportamento das funcionalidades de deleção e de consulta oferecida pelo framework Androjena.

5 CONCLUSÃO

Este trabalho mostra a factibilidade do uso de diversas estratégias que juntas são usadas para alcançar interoperabilidade de aplicações a nível de dados. Ao se obter interoperabilidade, informação pode ser gerenciada de forma mais inteligente e melhor controlada, possiblitando que cópias de dados possam se manter consistentes de forma mais simples. O uso de um repositório de dados unificado, juntando dados em comum de aplicações, permite que o controle de redundância de dados possa ser feito sem a necessidade de comunicação direta entre aplicações, evitando esforços na implementação de funcionalidades destinadas apenas à comunicação entre aplicações.

O uso de um modelo de dados flexível como o RDF permite que dados sejam consultados a partir de suas características e extendidos sem provocar incompatibilidades no modelo de dados utilizado por outras aplicações. O uso do RDF também permite que aplicações realizem tarefas com maior grau de autonomia do usuário, consultando dados existentes e apresentando conteúdo mais rico.

5.1 Dificuldades

5.1.1 Ontologias e Vocabulários

Ontologias e vocabulários, utilizados para dar significado aos esquemas URI (e portanto, significado aos dados que fazem uso dos esquemas) não são triviais de serem desenvolvidos e inconsistências entre relações podem surgir.

Um exemplo de inconsistência ou contradição pode ser observado na classificação de vírus como seres vivos, onde uma classe vírus pode ou não ser considerada uma subclasse de ser vivo. Inconsistências como a anteriormente mencionada prejudicam consultas que utilizam inferência para obtenção de resultados semanticamente válidos. Como ainda não existem muitas ontologias extensas, esses problemas ainda são críticos.

5.1.2 Evolução da Web

Apesar dos esforços para a padronização de tecnologias relacionadas à Web semântica, a grande maioria dos desenvolvedores não anotam páginas Web com dados compreensíveis por máquinas, prejudicando a evolução de aplicações que possam fazer uso desses dados. Aplicações que trabalham com modelos de dados como o RDF são mais complexos no aspecto de que a informação deve ser apresentada de duas formas diferentes para ser compreensível tanto por máquinas como por pessoas.

A área de consultas por padrões em grafos ainda não consegue obter performance comparável com os bancos de dados relacionais, capazes de processar em paralelo milhares de operações por segundo. Adicionalmente, sistemas de gerência de banco de

dados comerciais que trabalham sobre dados RDF existentes ainda não possuem maturidade o suficiente para serem utilizados em sistemas para fins comerciais.

5.2 Trabalhos Futuros

Trabalhos futuros imediatos devem ser focados na verificação prática da resolução dos problemas de redundância e inconsistência de dados através de um estudo de caso, indo além dos protótipos de aplicações desenvolvidas neste trabalho.

Apesar de não ter sido abordado com a devida importância, a implementação de políticas de privacidade e de mecanismos de controle de acesso a dados deve ser realizada, e de forma cautelosa, uma vez que o repositório de dados contém uma grande quantidade dados pessoais do usuário, tornando-se um alvo em potencial de ataques.

Como trabalhos futuros adicionais, a implementação de exemplos de adaptadores de domínio para aplicações nativas e de controle de acesso efetivo nos adaptadores e navegador concluiriam a parte de interoperabilidade entre aplicações Web e aplicações nativas para dispositivos móveis. Terminado esse enfoque, esforços podem ser dirigidos para estender a interoperabilidade para outros dispositivos e para os sensores do aparelho, completando toda a arquitetura proposta.

REFERÊNCIAS

INSTITUTE OF ELECTRICAL AND ELECTRONIC ENIGNEERING. **IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries**. New York, 1990.

BRODT, A.; NICKLAS, D. The TELAR mobile mashup platform for Nokia Internet Tablets. **EDBT '08: Proceedings of the 11th international conference on Extending database technology**. New York, NY, ACM Press, p. 700-704, Março 2008.

BRODT, A.; NICKLAS, D.; MITSCHANG, B. Deep integration of spatial query processing into native RDF triple stores. **Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems**. New York, NY, ACM Press, p. 33-42, Novembro 2010.

BRODT, A. et al. A mobile data management architecture for interoperability of resource and context data. **Proceedings of the 2011 Twelveth International Conference on Mobile Data Management**. [S.l], IEEE Computer Society, p. 1-6. Junho 2011.

BRICKLEY, D.; MILLER, L. **FOAF Vocabulary Specification 0.98**. [S.l.:s.n.], agosto 2010. Disponível em <http://xmlns.com/foaf/spec/>. Acesso em: jul. 2012.

BERNERS-LEE, T.; FIELDING, R.;MASINTER, L. **Uniform Resource Identifier (URI): Generic Syntax**: RFC 3986. [S.l.]: Internet Engineering Task Force, Network Working Group, 2005.

SEABORNE, A. et al. **SPARQL Update**. Julho 2008. Disponível em: <http://www.w3.org/Submission/SPARQL-Update/>. Acessado em: jul 2012.

PRUD'HOMMEAUX, E.; SEABORNE, A. **SPARQL Query Language for RDF**. Janeiro 2008. Disponível em: <http://www.w3.org/TR/rdf-sparql-query/>. Acessado em: jul 2012.

BRICKLEY, D.; GUHA, R. V.; MCBRIDE, B. **RDF Vocabulary Description Language 1.0: RDF Schema**. Fevereiro 2004. Disponível em <http://www.w3.org/TR/rdf-schema/>. Acesso em: jul. 2012.

Resource Description Framework. Fevereiro 2004. Disponível em <http://www.w3.org/RDF/>. Acesso em: jul. 2012.

HERMAN, I.; HAWKE, S.; PRUD'HOMMEAUX, E. **Semantic Web Activity**. Novembro 2011. Disponível em <http://www.w3.org/2001/sw/>. Acesso em: jul. 2012.

MEHTA, N. et al. **Indexed Database API**. Maio 2012. Disponível em <http://www.w3.org/TR/IndexedDB/>. Acesso em: jul. 2012.

HICKSON, I. **Web Storage**. Dezembro 2011. Disponível em <http://www.w3.org/TR/webstorage/>. Acesso em: jul. 2012.

LAFON, Y. **Web Services**. Maio 2008. Disponível em <http://www.w3.org/2002/ws/>. Acesso em: jul. 2012.

CHRISTENSEN, E. et al. **Web Services Description Language (WSDL) 1.1**. Março 2001. Disponível em <http://www.w3.org/TR/wsdl>. Acesso em: jul. 2012.

GUDGIN, M. et al. **SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)**. Abril 2007. Disponível em <http://www.w3.org/TR/soap12-part1/>. Acesso em: jul. 2012.

Telar DCCI. Disponível em <http://telardcci.garage.maemo.org/>. Acesso em: jul. 2012 .

Androjena. Disponível em <http://code.google.com/p/androjena/>. Acesso em: jul. 2012.

Apache Jena. Disponível em <http://jena.apache.org/>. Acesso em: jul. 2012.

RDFLib. Disponível em <https://github.com/RDFLib>. Acesso em: jul. 2012.

Pythonext. Disponível em <http://code.google.com/p/pythonext/>. Acesso em: jul. 2012.

XPCOM. Jul 2012. Disponível em <https://developer.mozilla.org/en/XPCOM>. Acesso em: jul. 2012.

What is an SSL certificate?. Disponível em <https://www.globalsign.com/ssl-information-center/what-is-an-ssl-certificate.html>. Acesso em: jul. 2012.

Android Philosophy. Disponível em <http://source.android.com/about/philosophy.html>. Acesso em: jul. 2012.

Android Reference. Disponível em <http://developer.android.com/reference/packages.html>. Acesso em: jul. 2012.

Intent. Disponível em <http://developer.android.com/reference/android/content/Intent.html>. Acesso em: jul. 2012.

Introducing JSON. Disponível em <http://www.json.org/>. Acesso em: jul. 2012.