JAVA EMERGENCY MANAGEMENT FRAMEWORK

JEMF

MANUAL

Programa de Pós-Graduação em Informática (PPGI)
Universidade Federal do Rio de Janeiro (UFRJ)

Marcus Machado

marcus.machado@ppgi.ufrj.br

2014

1 Gestão de Emergências

Emergências acontecem com certa frequência e muitas vezes resultam em perdas materiais e humanas. Em 2011, 332 desastres naturais mataram 30.773 pessoas e causou o maior dano econômico registrado até o momento: em torno de 366,100 dólares bilhões [15]. Tendo em vista essas ocorrências, muitos países se esforçam para colocar os planos nacionais de Gestão de Emergências (GE) em vigor, para resolver falhas nas estratégias de resposta existentes. A realização de estudos e desenvolvimento de novas tecnologias é uma parte essencial desses esforços.

A GE é caracterizada pelo seu aspecto colaborativo: envolve o trabalho de diferentes pessoas e organizações na resposta a situações de crise. Durante a resposta aos desastres, deve haver uma constante troca de informações entre a equipe de comando, responsável pelo planejamento e gerenciamento de tarefas de resposta e a equipe de operações, responsável pela execução das tarefas referidas, tais como o resgate de vítimas e combate a incêndios.

Geralmente, o dispositivo mais amplamente usado para essa finalidade é de rádio UHF. No entanto, os canais de rádio podem ser insuficientes para apoiar os processos de resposta e podem ficar congestionados. Essa limitação torna-se evidente quando a equipe de comando tem que coordenar emergências simultâneas. Além disso, vídeos, mapas, fotos e mensagens de texto são informações digitais relevantes que ajuda a organizar e coordenar a resposta de emergência. No entanto, não há nenhuma possibilidade de sua entrega através de um sistema de rádio [6].

Essas características destacam a complexidade da comunicação, colaboração, gerenciamento de informações e tomada de decisão em situações críticas. Sistemas de informação foram projetados para ajudar essas equipes através da integração da tecnologia moderna, tais como dispositivos móveis e vestíveis. Na década passada, a necessidade de fornecer dispositivos tecnológicos para as equipes de emergência melhorar o seu acesso à informação tornou-se evidente. Entre outras coisas, o uso de tecnologia ajuda com o desenvolvimento de um entendimento comum, entre as pessoas envolvidas, das tarefas a serem realizadas, tornando mais eficiente à resposta as emergências.

Vários estudos e projetos surgiram para apoiar a colaboração e aumentar a consciência situacional das equipes de resposta. Entre eles, alguns adotaram o uso de computação móvel [1,2,5,6,9,10,11,12,13,14,18,19] e discutem soluções de software para dispositivos móveis e experimentos, tendo em vista seus recursos e capacidades. Como estas soluções de software se tornam mais complexas, maior importância é colocada em técnicas de desenvolvimento de software para facilitar a sua produção.

A reutilização de software é um dos principais objetivos perseguidos pelos desenvolvedores e é essencial para simplificar o desenvolvimento de software neste domínio. Diante disso, o arcabouço Java Emergency Management Framework (JEMF) foi desenvolvido para contribuir na construção de sistemas móveis através da reutilização de software para apoiar a gestão de emergência.

2 Reuso de Software

Um arcabouço (framework) de software é normalmente composto por um conjunto de classes que implementa um projeto abstrato que aborda uma família de problemas relacionados [9]. Ele permite a reutilização de código e de projeto e contém um conjunto de classes que representam objetos de um domínio específico. A concepção de um sistema é geralmente descrita em termos de componentes, como eles interagem e colaboram entre si, as responsabilidades que cada um tem, e sua lógica de controle e fluxo de informação. Um arcabouço define estes elementos de modo a torná-lo fácil de usar componentes individuais, seguindo a estrutura completa de relacionamentos e padrões identificados em um domínio da aplicação. Quando um arcabouço é desenvolvido, os aspectos genéricos do domínio são capturados e diferentes aplicações podem ser desenvolvidas para fazer uso desses conceitos presentes no arcabouço, poupando tempo e esforço. Apenas os aspectos específicos da aplicação precisam ser modelados, uma vez que os requisitos genéricos já foram modelados e implementados.

De acordo com [5], os benefícios da utilização de um arcabouço incluem a modularidade, a reutilização e a extensibilidade. A modularidade acontece através de encapsulamento de implementações voláteis atrás de interfaces estáveis. Além disso, melhora a qualidade do software, devido ao impacto localizado de mudanças na concepção ou implementação. Isso reduz o esforço necessário para a compreensão e manutenção do software existente. Essas interfaces estáveis aumentam a reutilização através da definição de componentes genéricos que podem ser reutilizados para criar novas aplicações. Dessa forma, o conhecimento de domínio e esforço empregado por analistas experientes no desenvolvimento do arcabouço, evitando a necessidade de reconstruir e revalidar os requisitos de aplicações comuns e soluções de projeto recorrentes. Extensibilidade é fornecida por uma arquitetura adaptável que permite que aos aplicativos especializar suas classes e agregar novos recursos e serviços. Com o tempo, a reutilização de software leva a um aumento da produtividade e da confiabilidade do produto, e contribui para a redução de custos. Para permitir a reutilização, é necessária uma etapa anterior, onde as características comuns em um domínio de destino são identificadas, analisadas, projetadas e implementadas.

Um arcabouço geralmente usa vários padrões de projetos na sua construção, o que beneficia futuras expansões. Os elementos invariantes de um domínio são implementados no arcabouço e reutilizados nas instâncias. O arcabouço também reduz a complexidade técnica, uma vez que a arquitetura parcialmente implementado já foi definido e detalhes de implementação encapsulados [15]. Uma estrutura de apoio ao desenvolvimento de aplicações móveis com especificações genéricas permite a reutilização de software para a construção de soluções eficientes e de fácil manutenção, aprimorando o processo de desenvolvimento. O uso sistemático de arcabouços para modelagem, especificação e implementação de sistemas tem como objetivo melhorar o processo de desenvolvimento de software, garantindo menor esforço e maior qualidade do produto final. O arcabouço deverá permitir que os artefatos de um processo de criação de software (requisitos, a estrutura lógica, código, etc.) sejam reutilizados para o desenvolvimento de novos sistemas móveis.

3 Arcabouço JEMF

JEMF (Java Emergency Management Framework) busca contribuir com o desenvolvimento de uma arquitetura de software para tecnologia móvel e atuará na construção de sistemas para dispositivos móveis atuais, como smartphones e tablets. Através de artefatos e especificações apresentadas neste manual, contribuímos para orientar novos projetos que suportam os processos de resposta às emergências. Definimos alguns princípios essenciais para o desenvolvimento dos sistemas móveis e assim remover a complexidade de codificá-los.

JEMF visa melhorar a produtividade em novos projetos, trazendo os benefícios da reutilização, facilidade de manutenção e fornecer características detalhadas de modelagem e implementação. Então, espera-se contribuir com este arcabouço de software no desenvolvimento de novos sistemas por meio da reutilização de software.

3.1 Especificações Gerais

As especificações de projeto do JEMF foram obtidas basicamente a partir da análise das iniciativas anteriores abordadas na seção anterior. A partir da classificação das funcionalidades dos sistemas de informação mais relevantes, foram extraídas as funcionalidades genéricas e que constituem o arcabouço JEMF.

Requisitos.

Através dessas soluções computacionais é possível destacar alguns dos requisitos mais difundidos e que são necessários aos sistemas nesse domínio. A definição de requisitos específicos e funcionais é de competência de cada novo projeto a ser desenvolvido, portanto eles podem variar de acordo com diferentes características e objetivos das organizações dentro do domínio.

Requisitos Gerais.

- **Comunicação**: suporte ao compartilhamento de informação entre equipes de comando e operação. Acesso e coleção de informações sobre o desastre podem ajudar a organizar as operações de reposta, aumentar a colaboração e melhorar a segurança dos envolvidos.
- **Entrada e saída de dados**: A entrada de dados precisa ser simples e rápido para não tomar tempo hábil para realização de tarefas. A saída de dados precisa tornar fácil a visualização dos dados armazenados para que o usuário tenha uma melhor compreensão de dados importantes.
- **Integridade**: as tomadas de decisão dependem da integridade dos dados fornecidos. Portanto, o sistema deve representar a confiabilidade da fonte e a credibilidade da informação nas estratégias e meios de transferência dos dados.
- **Interoperabilidade**: o sistema deve ser aberto para troca de informações com outros sistemas. Assim como as informações devem ser exportáveis ou importáveis em um padrão que possa ser utilizado por outros sistemas.
- **Disponibilidade**: Os dados devem estar disponíveis a todos, em todos os lugares e a qualquer momento, permitindo aos usuários trabalharem também de forma assíncrona e oferecer opções para sincronizar com as equipes de operação e comando no momento necessário.

Requisitos Específicos.

- Gerenciar dados de **Emergências**: O arcabouço deve permitir a inclusão, a alteração, a exclusão e a consulta de emergências.
- Gerenciar dados de **Missões**: O arcabouço deve permitir a inclusão, a alteração, a exclusão e a consulta de missões.
- Gerenciar dados de **Tarefas**: O arcabouço deve permitir a inclusão, a alteração, a exclusão e a consulta de tarefas.
- Gerenciar dados de **Unidades de Saúde**: O arcabouço deve permitir a inclusão, a alteração, a exclusão e a consulta de unidades de saúde.
- Gerenciar dados de **Vítimas**: O arcabouço deve permitir a inclusão, a alteração, a exclusão e a consulta de vítimas.
- Gerenciar dados de **Testemunhas**: O arcabouço deve permitir a inclusão, a alteração, a exclusão e a consulta de testemunhas.
- Gerenciar dados de **Recursos**: O arcabouço deve permitir a inclusão, a alteração, a exclusão e a consulta de recursos.
- Gerenciar dados de **Pontos de Interesse**: O arcabouço deve permitir a inclusão, a alteração, a exclusão e a consulta de pontos de interesse.
- Gerenciar dados de **Mensagens**: O arcabouço deve permitir a inclusão, a alteração, a exclusão e a consulta de mensagens.
- Gerenciar dados de **Documentos Compartilhados**: O arcabouço deve permitir a inclusão, a alteração, a exclusão e a consulta de documentos compartilhados.
- Gerenciar dados de **Localizações**: O arcabouço deve permitir a inclusão, a alteração, a exclusão e a consulta de localização.

Atores.

Uma emergência pode demandar por atores de resposta com diferentes níveis de autoridade, funções e especialidades. Os sistemas de informação devem abranger funcionalidades para cada perfil de pessoa e para que elas possam exercer suas responsabilidades da melhor maneira possível com o apoio desses sistemas. As vítimas e testemunhas envolvidas nos incidentes também estão presentes nesse contexto, mas diferentemente dos respondedores, geralmente elas podem fazer parte do objetivo das tarefas de resgate e não são usuários dos sistemas (**Fig. 1**).

As características e responsabilidades desses atores que trabalharam com dispositivos móveis são descritas a seguir:

- **Comandante**: Ator responsável pela gerência da resposta às emergências. Ele define e atribui atividades para os demais usuários, concentra a maior parte das informações relevantes durante a resposta e realiza a tomada das decisões no local do incidente com a visão geral do contexto situacional. Exemplos: Comandante de incidente, oficial de polícia.
- **Respondedor**: Ator responsável pela execução das atividades de resposta ao incidente. O respondedor recebe as suas atividades atribuídas pelo comandante e as executa de acordo com sua especialidade. Ele também captura as informações contextuais in loco e compartilha com os demais atores. Seu principal objetivo é a estabilização do incidente e o resgate de vítimas. O respondedor engloba a maior parte das funções de resposta em campo. Exemplos: Bombeiro, policial, observador, motorista de veículo, piloto de helicóptero, etc.
- **Médico**: Ator responsável pela execução das atividades de atendimento médico as pessoas vitimadas. Ele detém a capacidade de avaliar e estabilizar a saúde das pessoas que

sofreram algum dano pelo incidente. Exemplos: Médico, paramédico, enfermeiro e outras profissões relacionadas à assistência em saúde pública.

- **Voluntário**: Ator responsável pela execução das atividades de resposta com menor prioridade ou risco. Ele oferece suporte ao comandante e respondedor na realização das ações que demandam maior quantidade de pessoas e podem auxiliar com conhecimentos especializados. Exemplos: Analista, técnico, empregados de companhias de luz, água e telecomunicações, agente de organizações não governamentais (ONGs), etc.
- **Vítima**: Representa a entidade que foi afetada pelo incidente, englobando desde uma pessoa até uma comunidade. As vítimas necessitam das ações de resgate do combatente, médico ou voluntário. Mas, geralmente não fazem parte do grupo de usuários dos sistemas de informação. Exemplo: Cidadão.
- **Testemunha**: Representa a entidade que testemunhou o incidente. Ela pode fornecer informações relevantes sobre o fato ocorrido para os demais atores responsáveis pela resposta à emergência. Da mesma forma que a vítima, geralmente não fazem parte do grupo de usuários dos sistemas de informação. Exemplos: Cidadão ou observador.

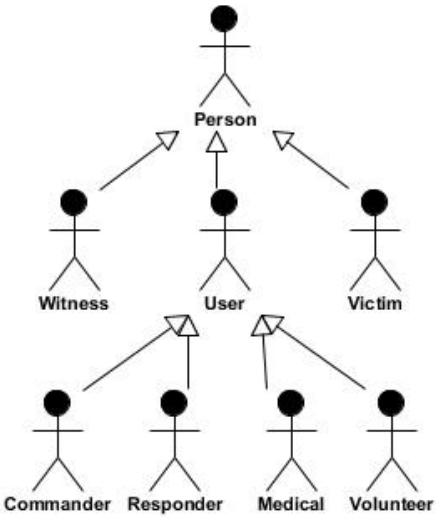

Fig. 1. Principais atores envolvidos em Gestão de Emergências.

Funcionalidades.

Dentre as diversas funções providas pelos projetos referenciados, selecionamos as funcionalidades mais relevantes que fazem parte do JEMF (**Fig. 2**). Elas representam um conjunto de ações fundamentais que os sistemas de informação em gestão de emergências devem apoiar para um gerenciamento e tomada de decisão adequada pelos atores nos atendimentos aos desastres.

A seguir, detalhamos o objetivo das funcionalidades para o gerenciamento dos dados que englobam as atividades de resposta, sendo que cada uma é composta pelas operações de inclusão, consulta, alteração e exclusão de dados (CRUD).

- **Gerenciar Emergência**: Funcionalidade que gerencia os dados do evento principal do desastre ocorrido, ou seja, serve para registrar qual o tipo, a intensidade, o local e a data da emergência que estará sendo tratada. Por exemplo, um incêndio florestal.
- **Gerenciar Missão**: Funcionalidade que gerencia os dados de macro atividades destinadas a um grupo de agentes que poderão ser registradas pelos gestores de crise. Por

exemplo, atividades de combate ao incêndio na região A, busca e salvamento de vítimas na região B, retirada de cidadãos da região de risco C, etc.

- **Gerenciar Tarefa**: Funcionalidade que gerencia os dados de atividades individuais de cada agente de resposta alocado numa missão. Por exemplo, agente 1 responsável pela atividade de rescaldo, agente 2 responsável pela atividade de estabilização de vítimas, agente 3 responsável pela operação de equipamentos ou veículos de resgate, etc.
- **Gerenciar Unidade de Saúde**: Funcionalidade que gerencia os dados das entidades de saúde que fazem parte da resposta à emergência. Por exemplo, registro de ambulâncias, tendas de atendimento e hospitais que poderão receber vítimas do desastre.
- **Gerenciar Vítima**: Funcionalidade que gerencia os dados inerentes a cada vítima socorrida no processo de resposta.
- **Gerenciar Testemunha**: Funcionalidade que gerencia os dados de possíveis testemunhas do evento ocorrido, permitindo adquirir informações que poderão ser importantes para a execução de atividades pelos agentes.
- **Gerenciar Recurso**: Funcionalidade que gerencia os dados dos equipamentos disponíveis para suporte das atividades realizadas pelos agentes em campo.
- **Gerenciar Ponto de Interesse**: Funcionalidade que gerencia os dados de localização de pontos estratégicos, como uma infraestrutura danificada pelo evento ou qualquer localidade que seja considerado relevante no processo de resposta.
- **Gerenciar Mensagem**: Funcionalidade que gerencia os dados que serão enviados e recebidos entre cada agente em campo, incluindo mensagens de texto, imagem ou áudio.
- **Gerenciar Documentos Compartilhados**: Funcionalidade que gerencia os dados que estarão compartilhados e considerados relevantes a todos os agentes em campo durante o período de resposta ao incidente, como planos de ação, documentos, etc.
- **Gerenciar Localização**: Funcionalidade que gerencia os dados que permite ao agente registrar a sua localização.

As funcionalidades estabelecidas para o JEMF estão organizadas por funcionalidades gerais e funcionalidades específicas. As funcionalidades gerais são destinadas a todos os atores envolvidos no processo de resposta, pois elas são funções fundamentais que os sistemas de emergências precisam apoiar, tais como a gestão de tarefas, recursos, pontos de interesse, mensagens e documentos compartilhados. As funções específicas são destinadas para cada ator de acordo com o seu nível de responsabilidade dentro da equipe de resposta. O comandante utiliza o gerenciamento de dados da emergência, missão e unidade de saúde. O respondedor usa o gerenciamento de dados da missão, vítima e testemunha. O médico usa o gerenciamento de dados da missão, vítima e unidade de saúde. O voluntário utiliza o gerenciamento de dados de testemunha.

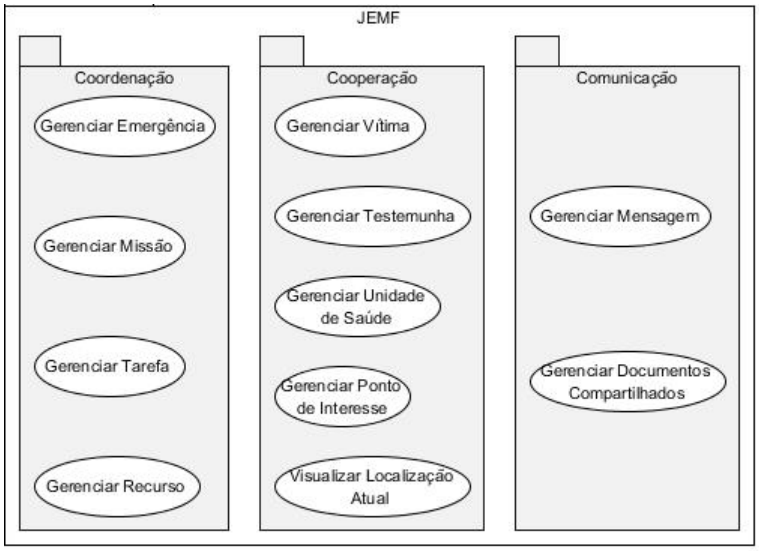

Fig. 2. Principais funcionalidades estabelecidas para o JEMF.

Classes.

Além dos requisitos, foram definidos diagramas de classe para a construção do arcabouço. Alguns estudos indicam que um arcabouço deve ser evoluído a partir de classes gerais e usar herança nos sistemas individuais [8, 17]. A herança permite aos desenvolvedores construir novas classes e alterar o código em um ambiente orientado a objeto, simplesmente herdando o comportamento desejado de uma classe existente e substituindo apenas os métodos que são diferentes na subclasse. A definição de muitas subclasses para especificar as diferenças de regras de negócio pode ser necessária, mas áreas funcionais chaves podem ser abstraídas fora e reutilizadas de forma conveniente.

Além disso, padrões de projeto foram aplicados para aumentar a quantidade de códigos reutilizáveis nas superclasses. JEMF é divido em dois pacotes: o Núcleo (Core) e o Móvel (Mobile). Ambos os pacotes podem ser personalizados porque todas as classes estão disponíveis para desenvolvedores, uma vez que este arcabouço é definido como código aberto.

Pacote Núcleo.

O pacote Núcleo é onde se concentram as classes fundamentais do arcabouço e representam entidades que envolvem uma resposta de emergência (**Fig. 3**). Espera-se que essas classes do núcleo sofram uma menor quantidade alterações, pois se tratam de classes com maior nível de abstração. A partir dessas classes poderão ser estendidas as classes concretas que formam o conjunto de funcionalidades descritas na seção anterior.

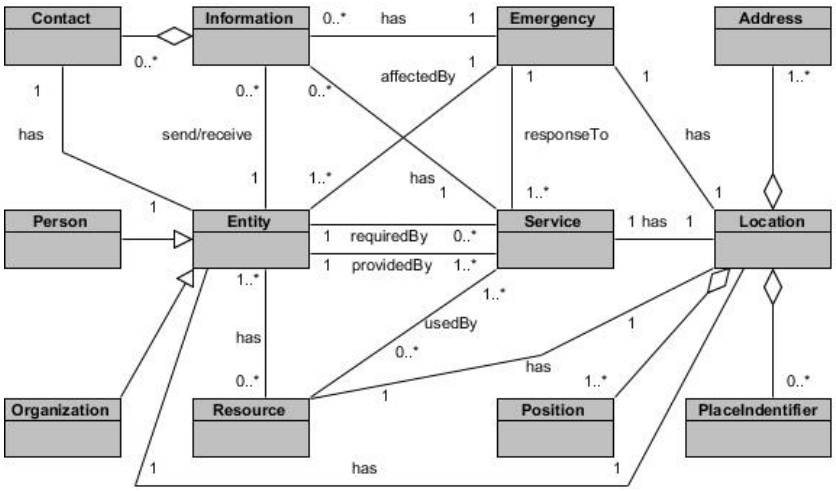

Fig. 3. Pacote Núcleo do JEMF destacando as principais classes (abstratas).

A **Fig. 3** apresenta as principais classes do pacote Núcleo, onde uma emergência (**Emergency**) necessita de serviços de resposta (**Service**). Esses serviços podem ser fornecidos por entidades de resposta as emergências (**Entity**) ou podem ser requisitados por entidades afetadas. Uma entidade pode ser uma organização, por exemplo, Corpo de Bombeiros, ou pode ser uma pessoa, por exemplo, um bombeiro (**Organization** ou **Person**).

Os serviços de resposta e entidades podem utilizar os recursos materiais disponíveis (**Resource**) durante a emergência. Assim como eles podem enviar ou receber informações entre si (**Information**). Uma entidade contém um dado para contato (**Contact**), por exemplo, número de telefone ou e-mail que podem ser usados para a troca de informações.

Uma emergência, um serviço, um recurso e uma entidade podem conter uma localização (**Location**). Essa localização pode ser formada por dados de logradouro, dados de posicionamento global ou dados para uma identificação personalizada (**Address**, **Position** e **PlaceIndentifier**).

A **Fig. 4** apresenta um conjunto de classes que se referem ao aspecto de coordenação. Além das classes principais já mencionadas, são destacadas algumas especializações das mesmas. Um serviço pode ser especializado em missão ou tarefa (**Mission** ou **Task**). Essa diferença é encontrada em alguns sistemas de informação que trabalham com a alocação de pessoa em serviços de resposta de formas distintas. A missão retrata a designação de um serviço para um grupo de pessoas, por exemplo, o grupo A de resgate de vítimas para o salvamento de vítimas numa emergência. Já a tarefa é retratada como um serviço para cada pessoa separadamente, por exemplo, o bombeiro B deve executar a atividade de apagar o incêndio e o bombeiro C deve verificar a reserva de água disponível para atuar na emergência.

Uma pessoa pode ser especializada em usuários (**User**) de um sistema de informação para resposta aos desastres. Dentre os principais usuários estão: (i) o comandante (**Commander**), representando a pessoa com maior nível de autoridade e que possui a responsabilidade de gestão de todos os serviços de resposta à emergência; (ii) o combatente (**Responder**), representando o agente que executa os serviços designados pelo comandante; (iii) o médico (**Medical**), representando o agente que possui conhecimentos na área médica para serviços específicos nessa área; (iv) o voluntário (**Volunteer**), representando o agente que realiza serviços com menor risco ou prioridade. Caso necessário, novas especializações de usuário podem ser criadas para atender outras funcionalidades. Uma tarefa pode ser designada a um

desses tipos de usuários. Entretanto, a missão é preciso ser designada a uma equipe de usuários (**Team**).

Um recurso pode ser especializado em equipamentos (**Equipment**), representando os materiais utilizados pelos agentes durante execução dos serviços, como mangueira, pá, machado, dentre outros. Caso necessário, novas especializações de recurso podem ser criadas, por exemplo, suprimentos ou veículos de resposta.

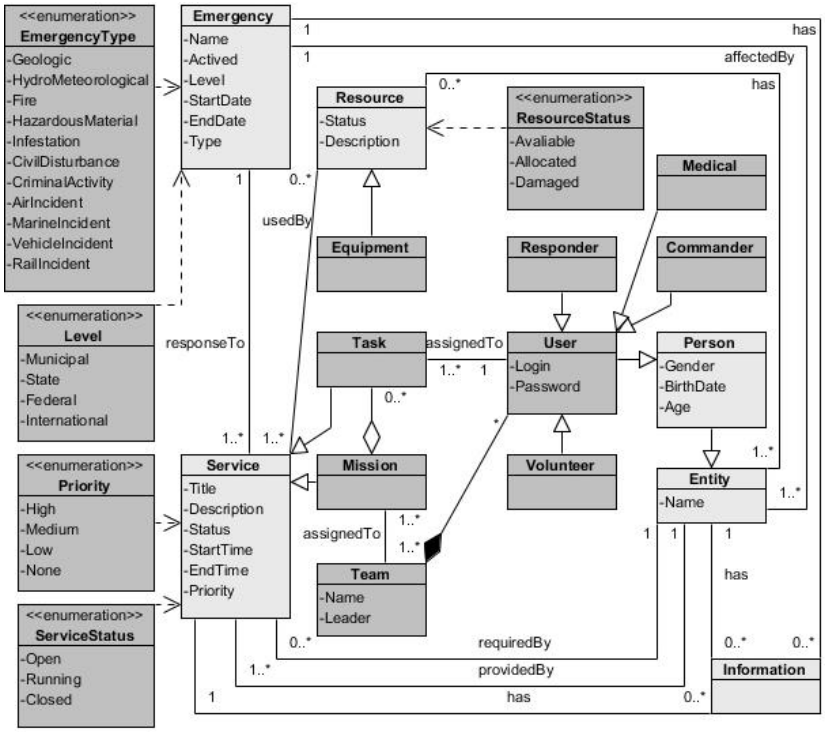

Fig. 4. Pacote Núcleo do JEMF destacando classes do aspecto de coordenação.

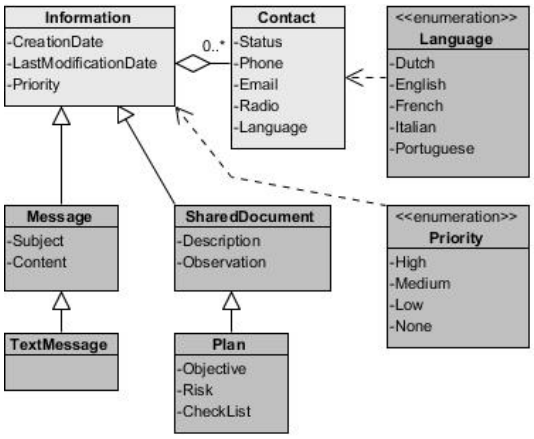

Fig. 5. Pacote Núcleo do JEMF destacando as classes para o aspecto de comunicação.

As **Fig. 5** e **Fig. 6** também destacam as especializações das classes principais descritas anteriormente. A **Fig. 5** apresenta um conjunto de classes que se referem ao aspecto de comunicação e a **Fig. 6** ao aspecto de cooperação.

Na **Fig. 5** é apresentada duas especializações para a classe de informação, mensagem (**Message**) e documentos compartilhados (**SharedDocument**). A classe mensagem representa os dados que os agentes podem trocar com outros agentes ou organizações. A classe mensagem de texto (**TextMessage**) representa uma especialização onde contém as propriedades para esse tipo de mensagem e a classe plano de resposta (**Plan**) contém propriedades específicas que podem ser compartilhados entre agentes numa situação de resposta.

Caso necessário, novas especializações de mensagem podem ser elaboradas, como mensagens para áudio ou vídeo. Já para a classe de documentos compartilhados, podem-se estender novas classes com outras propriedades, por exemplo, relatórios.

Na **Fig. 6** são evidenciadas outras duas especializações para a classe pessoa. A primeira é a vítima (**Victim**), representando as pessoas que foram afetadas pela emergência e necessitam de alguma assistência pelos agentes de resposta. A segunda é a testemunha (**Witness**), representando as pessoas que presenciaram e observaram a ocorrência da emergência e que podem informar dados relevantes sobre o evento.

Na **Fig. 6** também são observadas duas especializações para a classe organização, onde sua característica principal é definida em função da sua perspectiva ao desastre. A classe unidade de saúde (**HealthCareUnit**) representa as entidades que recebem as vítimas, envolvendo os hospitais ou posto de saúde no sentido de resgatar vítimas e atenuação das consequências do desastre na parte médica. A classe organização afetada (**AffectedOrganization**) representa as entidades que sofreram avarias ou consequências de um desastre ocorrido, envolvendo qualquer estrutura física, como edifícios ou casas. As classes unidade de saúde e organização afetada podem compor um ponto de interesse (**InterestPoint**), especializando a classe localização.

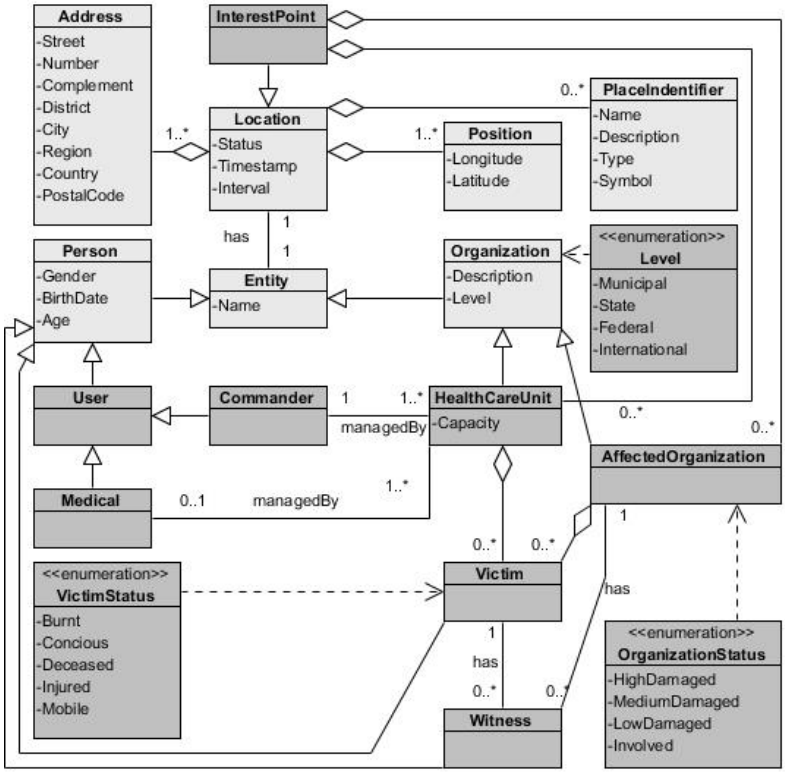

Fig. 6. Pacote Núcleo do JEMF destacando as classes para o aspecto de cooperação.

Pacote Móvel.

O pacote Móvel é destinado às classes concretas que estão atreladas com as propriedades disponibilizadas pelo sistema operacional móvel, como padrões, componentes e recursos. Dessa maneira, elas poderão ser reutilizadas para integrar a arquitetura de software de um novo sistema móvel. Este pacote contém as operações para apoiar a gestão de dados e permite que os desenvolvedores modifiquem e incrementem métodos oferecidos por essas classes concretas ou agreguem com outras classes da plataforma Android.

As **Fig. 7**, **Fig. 8** e **Fig. 9** apresentam as principais classes do pacote Móvel, focando como exemplo a funcionalidade Gerenciar Emergência. Entretanto, a estrutura e as relações entre as classes genéricas desse pacote para as outras funcionalidades são basicamente as mesmas. Nas figuras são destacados também os padrões de projetos aplicados para apoiar a reutilização de software (notas de observação na cor verde).

Na **Fig. 7** são destacadas na parte superior duas classes nativas da API Android - *Activity* e *Fragment* - que representam como uma aplicação cliente faz uso do arcabouço JEMF. Uma *Activity* é uma classe focada na interação com o usuário do sistema, sendo essas as classes que servem basicamente como janelas (telas) na interface do usuário e gerencia os componentes visuais [26]. Uma classe *Fragment* representa uma parte da interface ou um comportamento que compõem uma *Activity* [27]. Dessa forma, é possível combinar vários *Fragments* em uma única *Activity* para criar uma interface mais completa, ou seja, um *Fragment* funciona como uma seção modular (parte) de uma *Activity*, conforme detalhado na referência [28]. No Capítulo 4 será abordada a codificação para instanciar o arcabouço JEMF por uma classe *Activity*, demonstrando como são realizadas as operações mais importantes através de exemplos. Ainda na **Fig. 7**, são destacadas três principais classes e que serão detalhadas a seguir. Mas, será abordado primeiramente sobre alguns padrões de projetos apontados pelas notas de observação (cor verde).

O primeiro padrão a ser brevemente descrito é o *Abstract Factory*. O principal objetivo dele é oferecer uma interface para a criação de uma família de objetos relacionados, sem especificar explicitamente suas classes para uma aplicação cliente. O *Abstract Factory* determina qual é o tipo do objeto concreto que se desejar criar, e depois de criá-lo, retorna um ponteiro abstrato desse objeto para o cliente. O fato de o *Abstract Factory* retornar um ponteiro do objeto criado significa que o cliente não tem conhecimento do tipo real do objeto. Isto implica que não há nenhuma necessidade para a inclusão de quaisquer declarações de classe relacionadas com o tipo concreto na aplicação cliente, pois ela acessa em todos os momentos o ponteiro abstrato. Os objetos do tipo concreto, criados pela fábrica, são acessados pela aplicação cliente apenas através da interface abstrata.

O segundo padrão de projeto é o *Singleton*. Ele é responsável por fornecer um ponto global de acesso a um objeto, certificando-se de que apenas seja criada uma instância da sua classe. Neste caso, a mesma instância pode ser usada por várias vezes e em vários locais, sendo impossível invocar diretamente o construtor da classe.

A classe **EmergencyManagementFactory** tem por objetivo oferecer uma interface abstrata para acessar todas as funcionalidades disponibilizadas pelo arcabouço. Nela é implementado o padrão de projeto *Abstract Factory* que administra a criação de todos os objetos concretos do arcabouço. Assim como é implementado o padrão *Singleton* para que essa classe seja o ponto global de acesso ao arcabouço.

A classe **OptinalFactory** na versão atual do arcabouço é considerada apenas um gancho para as versões futuras do mesmo, onde espera-se que sejam criadas funções para acessar a API Android para manipulação de dados dos recursos de um dispositivo móvel, como por exemplo os seus sensores, para apoiar novas funcionalidades aos sistemas móveis. Espera-

se também que essa classe possa futuramente gerenciar, se necessário, o acesso às bibliotecas desenvolvidas por terceiros.

A classe **MainFactory** tem por finalidade gerenciar a criação de objetos concretos para as funcionalidades especificadas na seção anterior. Logo, a partir da notificação do tipo da funcionalidade pela aplicação cliente (Emergência, Missão, etc.), o arcabouço poderá manipular os seus objetos concretos para atender as funções requisitadas por ela, como salvar, excluir e carregar dados. Além disso, minimiza a declaração de classes do arcabouço na aplicação cliente para realizar estas mesmas funções.

A classe **FeatureHolder** é uma classe do tipo *Enumaration* que mantém os tipos de todas as funcionalidades disponibilizadas no arcabouço. Assim, por meio dessa classe o arcabouço identifica qual é a funcionalidade que está em operação, segundo a requisição da aplicação cliente.

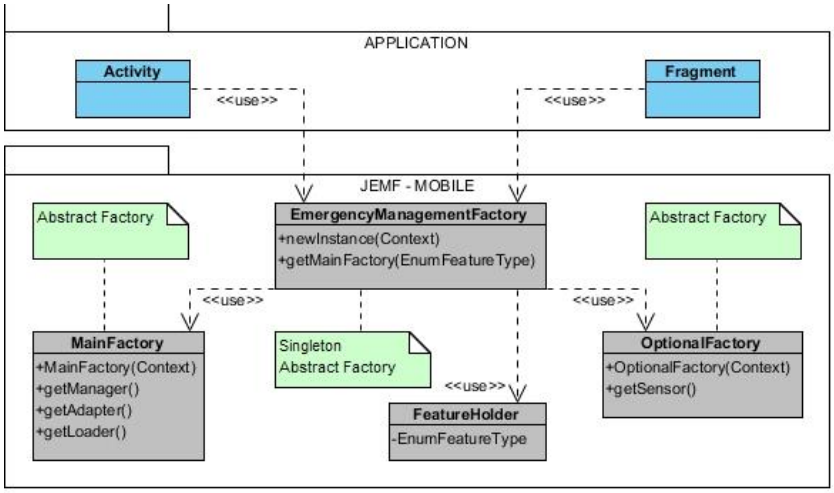

Fig. 7. Pacote Móvel do JEMF destacando as superclasses e os padrões de projetos (Parte 1).

O terceiro padrão de projeto é o *Factory Method*. Ele estabelece uma interface para a criação de objetos, mas deixa que as subclasses decidam qual a classe será instanciada, referindo-se ao objeto recém-criado através de uma interface comum.

O quarto padrão de projeto é o *Template Method*. Ele define o esqueleto de um algoritmo de uma operação, repassando algumas partes do algoritmo para as subclasses. Dessa maneira, ele permite que as subclasses redefinam determinadas etapas do algoritmo sem deixá-los mudar ou replicar toda a estrutura do algoritmo.

O quinto padrão de projeto é o *Adatpter*. Como qualquer adaptador no mundo real, ele é usado para ser uma interface, uma ponte entre dois objetos que não poderiam de outra forma trabalhar em conjunto. Ele converte a interface de uma classe para outra interface, permitindo que classes com interfaces diferentes e incompatíveis se comuniquem.

Na **Fig. 8**, a classe **ManagerFactory** tem por função gerenciar a instanciação dos objetos da classe **Manager** de forma a estabelecer somente uma interface para tal operação. A classe **Manager** é uma das principais classes de todo o arcabouço, pois ela define as funções fundamentais para inserir, alterar, excluir e carregar dados do banco de dados. As partes genéricas de todas essas funções foram codificadas nessa classe, de maneira que as partes variantes para atender separadamente cada funcionalidade estabelecida (Emergência, Missão, etc.) foram criadas interfaces que permitem a redefinição para atender as características individuais.

A classe **AdapterFactory** tem por objetivo administrar a criação dos objetos através de interfaces para as classes **CustomCursorAdapter** e **CustomLoader**. A classe **CustomCursorAdapter** constitui uma nova interface para a classe da API Android: *CursorAdapter* [29]. O *CursorAdapter* é um adaptador que expõe os dados de um *Cursor* [30], que mantem os dados carregados do banco de dados, para um componente *ListView* [31], responsável por exibir esses dados em uma lista de itens com rolagem vertical.

A classe **CustomLoader** é uma extensão da classe da API Android: *Loader* [32]. O *Loader* executa o carregamento assíncrono de dados, e enquanto está ativo, ele deve monitorar a fonte de seus dados e entregar novos resultados quando o conteúdo é alterado. Portanto, as classes **CustomCursorAdapter** e **CustomLoader** necessitam trabalhar em conjunto para preservar os dados carregados sempre atualizados na aplicação cliente de maneira apropriada.

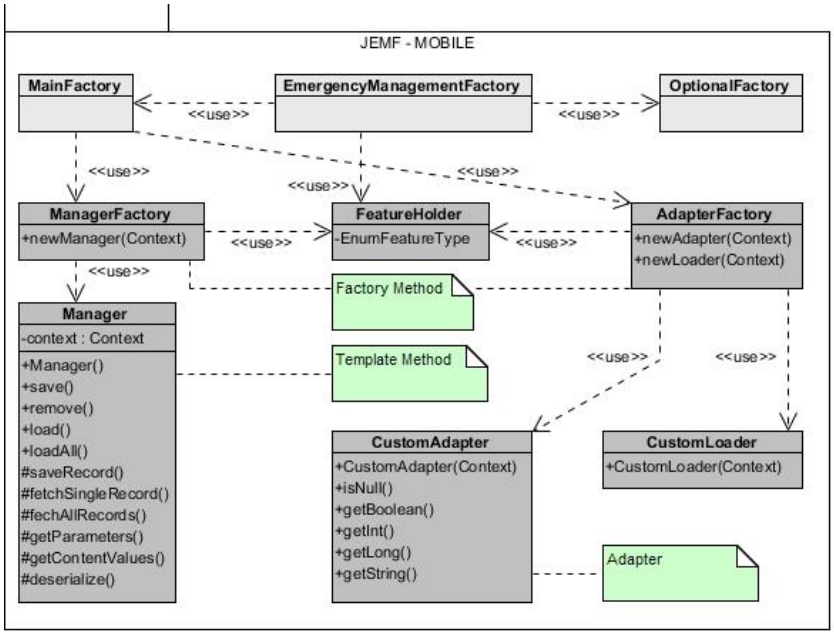

Fig. 8. Pacote Móvel do JEMF destacando as superclasses e os padrões de projetos (Parte 2).

Na **Fig. 9**, a classe **CustomProvider** é uma especialização da classe *ContentProvider* da API do Android [33]. O C*ontentProvider* encapsula os dados e através de uma interface única os compartilha para uma ou mais aplicações cliente. Por exemplo, os dados de contatos são utilizados por várias aplicações e devem ser armazenado por um único *ContentProvider*. A classe **DBHelper** é uma especialização da classe *SQLiteOpenHelper* [34]. O *SQLiteOpenHelper* é uma classe auxiliar para gerenciar a criação e versão do banco de dados. A classe **DBStatement** também é uma classe auxiliar e que organiza todas as declarações das consultas (*queries*) para criação e exclusão das tabelas do banco de dados. Já a classe **ProviderStatement** tem uma função similar, porém é para a declaração de propriedades para o **CustomProvider**.

Na **Fig. 9** é destacado um conjunto de classes que compõem a funcionalidade Gerenciar Emergência: **EmergencyManager**, **EmergencyAdapter** e **EmergencyLoader**. Essas classes herdam das classes genéricas já mencionadas (*Manager*, *CustomAdapter* e *CustomLoader*) as propriedades e funções básicas e torna-se necessário apenas codificar as partes específicas para trabalhar com os dados da emergência. Além disso, a classe **Emergency-**

View detém a codificação de uma interface básica com os componentes visuais já pré-estabelecidos para a funcionalidade Gerenciar Emergência.

Deste modo, para adicionar uma nova funcionalidade é preciso criar respectivamente novas especializações dessas superclasses em destaque na **Fig. 9**. Caso necessário, todas as classes são passíveis de personalização para atender novos requisitos.

Na parte inferior da **Fig. 9** é apresentado a classe **AbstractEmergency** do pacote Núcleo. A funcionalidade Gerenciar Emergência demanda que seja criada uma classe do tipo *JavaBean* [35]. Nesse caso, é definida a classe **Emergency** a partir da concretização da classe **AbstractEmergency** com o objetivo de estabelecer as propriedades da emergência e suas interfaces através dos métodos *getter* e *setter* para trabalhar de forma adequada com a plataforma Android.

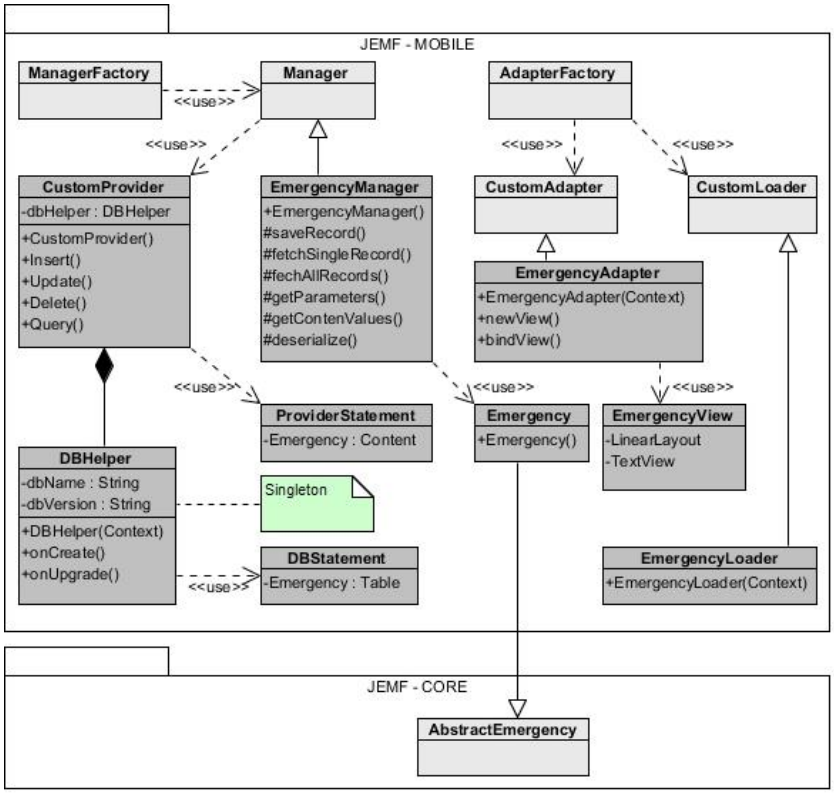

Fig. 9. Pacote Móvel do JEMF destacando a funcionalidade Gerenciar Emergência.

3.2 Plataforma

JEMF foi desenvolvido para a plataforma Android, que é um dos sistemas operacionais móveis mais difundidos hoje em dia. A plataforma Android é uma coleção de software que inclui um sistema operacional e um número de bibliotecas de alto nível que simplificam a tarefa de se comunicar com o sistema operacional. Ele também inclui vários aplicativos que os usuários de smartphones utilizam, como um telefone, cliente de e-mail, gerenciador de contatos, mapas, um navegador da web, um calendário e assim por diante.

Tudo no ambiente de desenvolvimento Android, bem como todos os aplicativos incluídos, pode ser programado com uma combinação de códigos escritos em Java e arquivos XML, graças a SDK do Android. Este SDK traduz em tempo de execução o código Java e

o XML escritos pelo desenvolvedor em uma linguagem que o sistema operacional e o dispositivo entendem.

A base sobre a qual é construído o Android foi cuidadosamente codificada para o Linux, um sistema operacional que raramente falha. O Linux e seus serviços gerenciam o telefone físico e dão acesso aos aplicativos suas principais características e funções, como tela sensível ao toque, memória, dados, segurança, vários receptores e transmissores de rede, câmera e muito mais. Porém, o Linux não faz tudo sozinho. O Android tem um número de bibliotecas que fornecem outros serviços para trabalhar com gráficos, formatos de áudio e vídeo, e funções personalizadas do mais amplo uso para as aplicações.

4 Projeto de Aplicativo

Nesta seção são abordados os pré-requisitos e os detalhes para a implementação de sistemas de informação móveis em conjunto com o JEMF para o Android.

4.1 Ambiente de desenvolvimento

O ambiente de desenvolvimento (*IDE*) recomendado para a implementação de sistemas com o uso do JEMF é o Android SDK [20] em conjunto com o **ADT Bundle** [21]. O ADT (*Android Developer Tools*) é um plug-in que fornece um conjunto de ferramentas integradas com a ferramenta Eclipse para desenvolvimento em linguagem Java. Ele oferece acesso a muitos recursos que ajudam todo o desenvolvimento de aplicativos Android, principalmente para a codificação e a prototipagem de interfaces de usuário. Assim, o ADT constitui-se como um ambiente de desenvolvimento bem estabelecido por oferecer recursos específicos para construção de aplicativos de forma rápida e organizada.

O download do ADT Bundle pode ser realizado no seguinte endereço:
http://developer.android.com/sdk/index.html

4.2 API Level

A API Level [22] é um identificador da revisão do arcabouço API oferecido para uma versão da plataforma Android. A plataforma Android fornece um arcabouço API onde os aplicativos podem usar para interagir com o sistema Android. Cada versão sucessiva da plataforma Android pode incluir atualizações para o arcabouço API que ele oferece. As atualizações para o arcabouço API são projetadas para que a nova API permaneça compatível com versões anteriores da API. Ou seja, a maioria das alterações na API é aditiva e introduzem novas ou substituem funcionalidades. Como partes da API são atualizadas, as partes mais antigas se tornam obsoletas, mas não são removidas, para que aplicativos existentes ainda pode usá-los. Esse arcabouço API é composto por:

- Um conjunto de pacotes e classes;
- Um conjunto de elementos e atributos XML para declarar um arquivo de manifesto;
- Um conjunto de elementos e atributos XML para declarar e acessar recursos;
- Um conjunto de permissões que os aplicativos podem solicitar;

O JEMF está configurado para o funcionamento a partir da **API Level 15**, Android versões **4.0.3** - **4.0.4**, codinome **ICE CREAM SANDWICH**. Não foram realizados testes de regressão da API Level, portanto não é possível afirmar o correto funcionamento em versões anteriores.

Na **Fig. 10** a seguir é evidenciado que a maior parte dos dispositivos atualmente utiliza o Android a partir versão 4.0.3, o que possibilita o JEMF operar em qualquer versão superior [23].

Version	Codename	API	Distribution
2.2	Froyo	8	0.7%
2.3.3 - 2.3.7	Gingerbread	10	13.6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	10.6%
4.1.x	Jelly Bean	16	26.5%
4.2.x		17	19.8%
4.3		18	7.9%
4.4	KitKat	19	20.9%

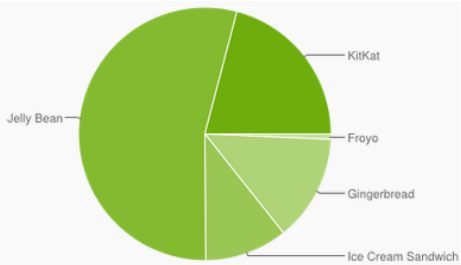

Fig. 10. Número de dispositivos que executam uma determinada versão da plataforma Android.

4.3 Configurando o ADT Bundle no Windows

Para a configuração do ADT Bundle no Windows é necessário seguir os seguintes passos:

1. Download do Android SDK ADT bundle (certifique-se de selecionar uma versão para coincidir com a plataforma do computador, 32/64-bit);
2. Descompacte o arquivo zip em um diretório de sua preferência;
3. Abra o diretório "*adt-bundle-<plataforma_so>/eclipse/*" e execute a aplicação "*eclipse.exe*";
4. A partir do menu "*Windows*" no Eclipse, selecione "*Android SDK Manager*";
5. A versão atual do JEMF exige que seja instalada a API versão 14 da "*SDK Platform*" e "*Google APIs*" ou superior. Então, marque a opção ao lado dos itens exibidos no Eclipse. Se você também deseja usar um dispositivo virtual (para emulação), adicione também o "*ARM EABI System Image*". Selecione os seguintes pacotes:

   - Android 4.0.3 (API 15)
     - SDK Platform;
     - ARM EABI v7a System Image;
     - Google APIs;

6. Clique no botão "*Install 3 packages*";
7. Selecione a opção "*Accept All*" e depois clique no botão "*Install*";
8. Reinicie o Eclipse.

4.4 Repositório SVN e o plug-in Subversive

Apache Subversion (SVN) [24] é o sistema de controle de versão usado para manter as versões atuais e históricas de arquivos tais como o código-fonte e documentação deste projeto. Esse sistema permite rastrear as alterações feitas durante o desenvolvimento do software com um controle do histórico do projeto, facilitando analisar e resgatar as versões antigas dos arquivos. Ele também permite separar o projeto em linhas de desenvolvimento para implementações paralelas, apoiando o trabalho em equipe sobre o mesmo conjunto de documentos ao mesmo tempo.

O projeto **Subversive** [25] é um plug-in do Eclipse que fornece suporte para trabalhar com projetos armazenados em repositórios SVN. Portanto, é necessário a sua instalação para acessar o código-fonte e documentação do arcabouço JEMF.

O download do Subversive pode ser realizado no seguinte endereço:

http://www.eclipse.org/subversive/downloads.php

As instruções para a instalação apropriada do Subversive se encontram no seguinte endereço:

http://www.eclipse.org/subversive/documentation/gettingStarted/aboutSubversive/install.php

4.5 Configurando o arcabouço JEMF no Eclipse

Após a instalação do Subversive, o próximo passo é realizar o download do arcabouço JEMF diretamente para o Workspace no Eclipse. O código-fonte e a documentação do arcabouço JEMF estão disponíveis no seguinte endereço:

https://code.google.com/p/jemf/

Uma vez que o Eclipse estiver aberto, selecione o menu principal para criar um novo projeto: "*File* > *New* > *Project*". Na caixa de diálogo "*New Project*", selecione a opção para fazer o download de um projeto SVN na lista de assistentes para informar ao Eclipse o tipo de projeto deseja ser importado, conforme mostrado na **Fig. 11**. Clique no botão "*Next*" para continuar.

Nesta caixa de diálogo "*Checkout from SVN*", é preciso que seja escolhida a opção "*Create a new repository location*" para que se possa inserir o endereço Web onde o projeto está armazenado e clique no botão "*Next*" para continuar (**Fig. 12**). Em seguida, é apresentado o campo "*Url*" onde se deve inserir o endereço SVN do JEMF abaixo (**Fig. 13**). Clique no botão "*Next*" para continuar.

http://jemf.googlecode.com/svn/

Por fim, um conjunto de diretórios do projeto é exibido na qual se deve selecionar a pasta "*Trunk*". Essa pasta contém os arquivos mais atuais do projeto e, portanto selecione as pastas para que sejam adicionadas no Eclipse (**Fig. 14**). Para terminar o processo, clique no botão "*Finish*" e aguarde o download dos arquivos para o Gerenciador de Pacote ("*Package Explorer*"). Após o download, o JEMF estará habilitado para o desenvolvimento no computador.

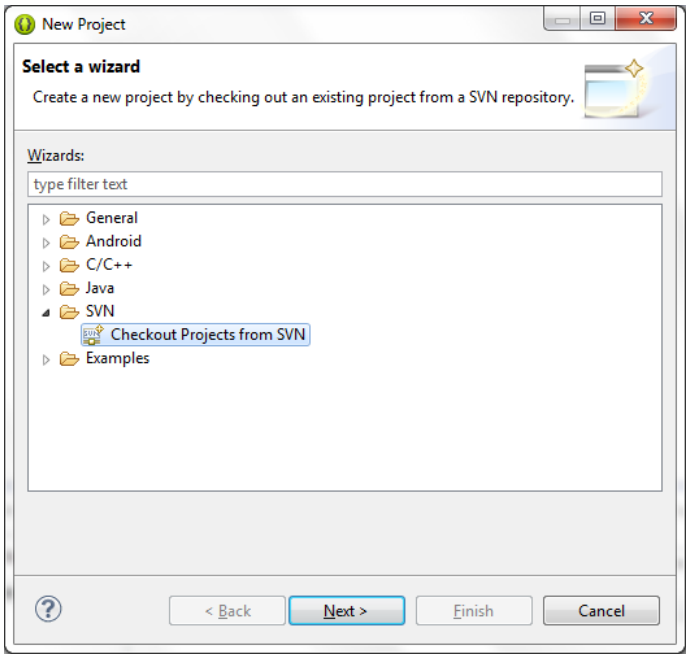

Fig. 11. Caixa de diálogo para acessar o projeto no SVN.

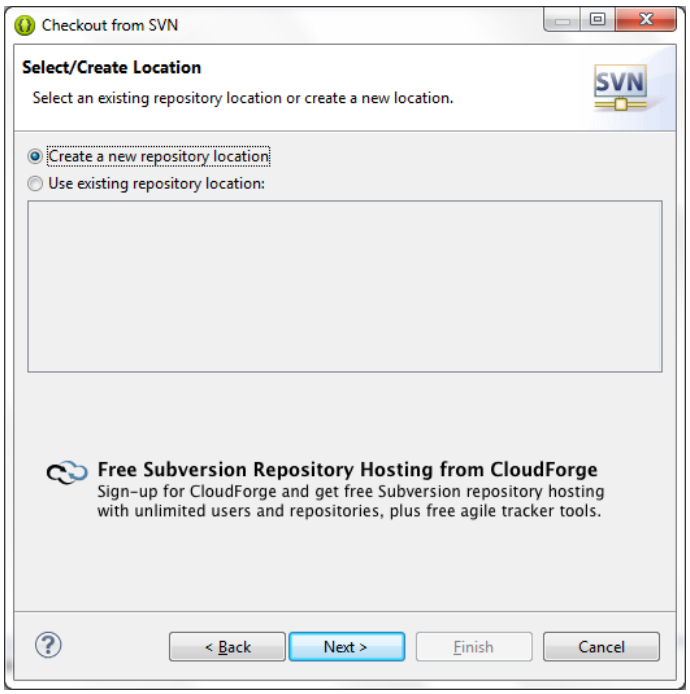

Fig. 12. Caixa de diálogo para criação de um link para o projeto no SVN.

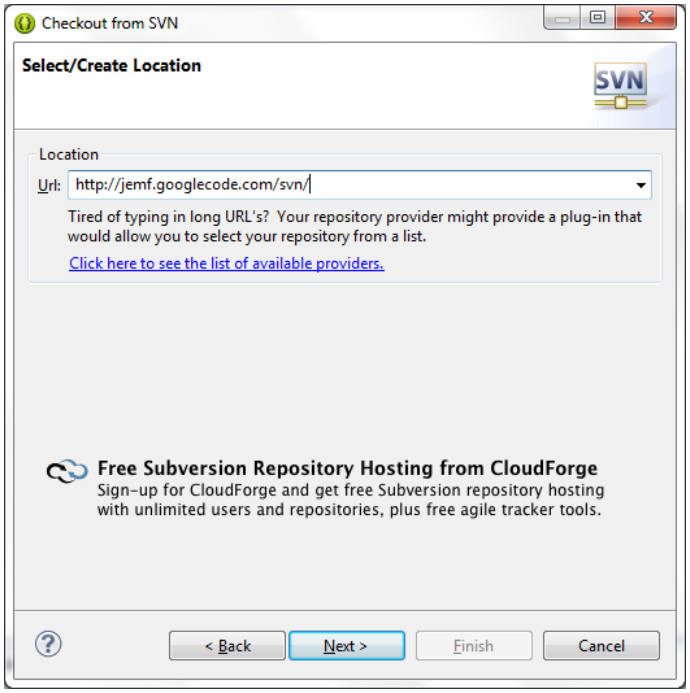

Fig. 13. Caixa de diálogo para inserção do link de um projeto no SVN.

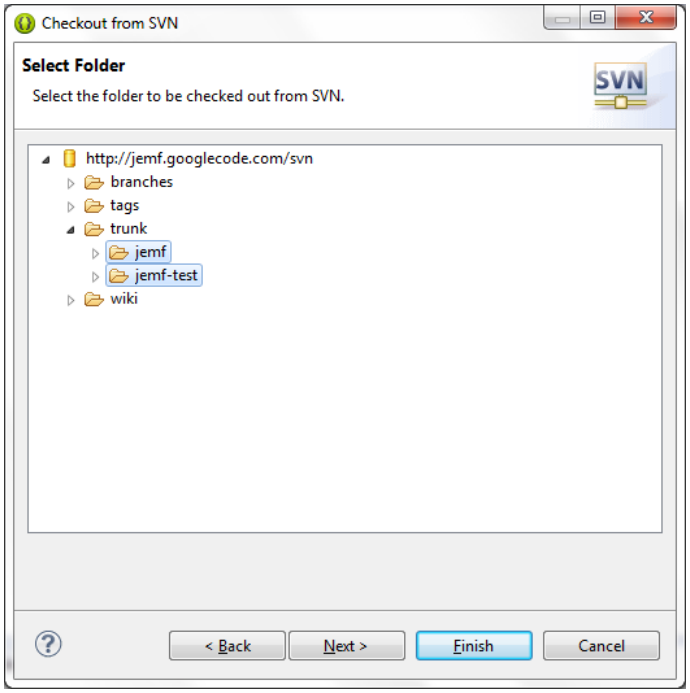

Fig. 14. Caixa de diálogo para seleção do arcabouço JEMF no SVN.

4.6 Criando um novo Projeto de Aplicativo

Uma vez que o Eclipse estiver aberto, selecione o menu principal para criar um novo projeto: "*File* > *New* > *Project*". Na caixa de diálogo "*New Project*", selecione o projeto Android na lista de assistentes para informar ao Eclipse o tipo de projeto deseja ser criado, conforme mostrado na **Fig. 15**. Clique no botão "*Next*" para continuar.

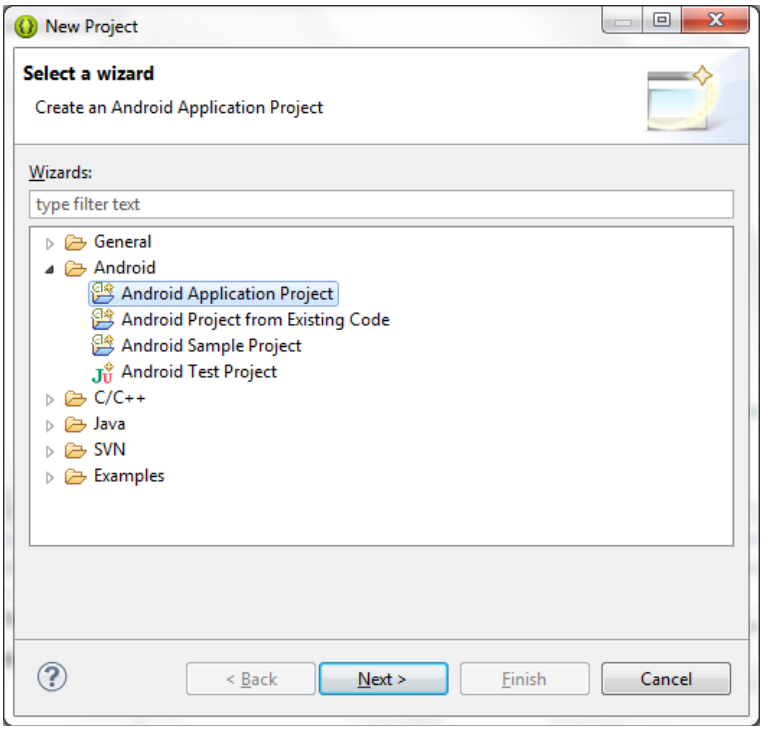

Fig. 15. Caixa de diálogo para criação de um novo projeto Android.

Nesta caixa de diálogo para um novo projeto Android, é preciso que sejam especificados todos os tipos de propriedades importantes para o aplicativo (**Fig. 16**). Essas propriedades permitem que seja informado onde se deseja criar o novo aplicativo, o local onde o código Java e os arquivos XML serão gravados pelo IDE. As propriedades estão detalhadas a seguir:

- **Application Name**: Este é o nome do aplicativo, ou seja, é o nome que aparecerá na barra de título do aplicativo quando ele é executado. Nesse exemplo, o nome do aplicativo é: "*HelloWorldAndroid*".
- **Project Name**: Este é o nome da pasta que mantém os arquivos e subpastas de um novo aplicativo. Nesse exemplo, esta pasta possui o mesmo nome do aplicativo: "*HelloWorldAndroid*".
- **Package Name**: Este é o nome para o pacote que contém os códigos Java que o aplicativo usa. Nesse exemplo, o nome do pacote é: "*myapp.helloworld*".
- **Minimum Required SDK**: Este valor refere-se à seleção da API Level mínima para a construção do aplicativo. Nesse exemplo, o valor definido como API 15 (Android versão 4.0.3), condizendo com a API Level estabelecida para o JEMF.
- **Target SDK**: Este valor refere-se à seleção da API Level principal na qual o aplicativo será construído. Nesse exemplo, o valor definido como API 18 (Android versão 4.3), porém esse valor poderá ser qualquer versão igual ou acima da API Level mínima. Essa

versão contém os recursos para construir a maioria dos aplicativos que funcionam em todos os smartphones Android atuais.

- **Compile With**: Este valor refere-se à seleção da API Level na qual o aplicativo será compilado. Nesse exemplo, o valor definido como API 18 (Android versão 4.3), porém esse valor poderá ser qualquer versão igual ou acima da API Level mínima.
- **Theme**: Este é o nome do tema que o aplicativo adotará em sua aparência, uniformizando os layouts para os componentes do mesmo.

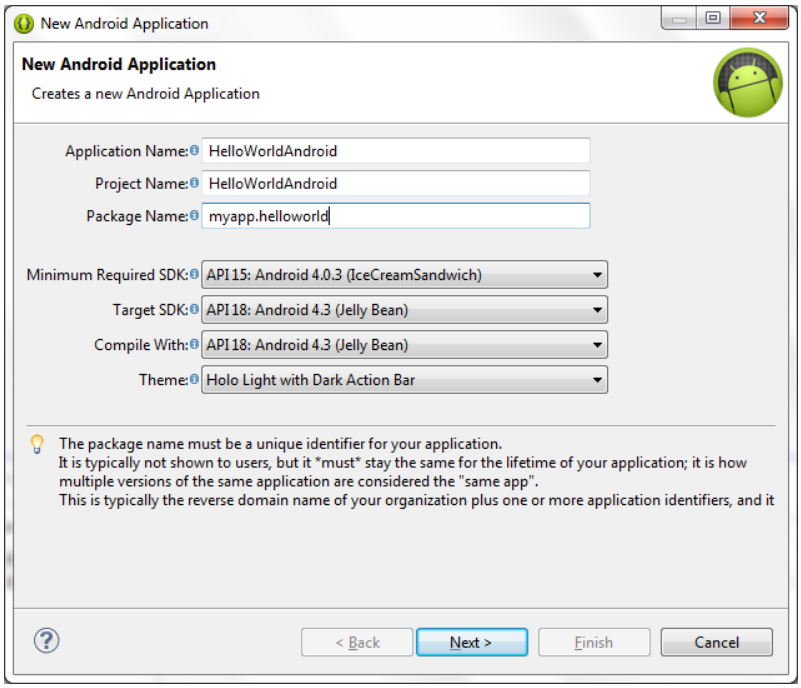

Fig. 16. Caixa de diálogo para atribuição de propriedades do novo projeto Android.

Após o preenchimento dessa parte, clique no botão "*Next*" para continuar. Nesta caixa de diálogo também, é possível configurar outras propriedades, como o ícone do aplicativo (**Fig. 18**), o nome do arquivo da tela principal e o caminho onde será salvo o aplicativo localmente no computador (**Fig. 17** e **Fig. 19**).

A configuração das propriedades nas telas "*Configure Project*", "*Configure Laucher Icon*", "*Configure Activity*" e "*Select kind of Activity*" da caixa de diálogo NÃO requerem uma edição em especial (respectivamente as figuras **Fig. 17**, **Fig. 18**, **Fig. 19** e **Fig. 20**). Portanto, clique no botão "*Next*" para continuar até a conclusão com o botão "*Finish*". O Eclipse irá criar os arquivos do projeto "*HelloWorldAndroid*".

- **Create Activity**: Esta opção cria uma classe *Activity* para o aplicativo de forma automática. Essa classe Java é uma coleção de códigos que controla a tela principal (interface do usuário) do aplicativo.
- **Create Project in Worspace**: Esta opção permite verificar a estrutura de pastas que o Eclipse usa para o projeto, mantenha esta opção marcada.
- **Activity Name**: Este é o nome para a *Activity* que contém os códigos Java da tela principal do aplicativo. Nesse exemplo, o nome do arquivo é: "*MainActivity*".

- **Layout Name**: Este é o nome para o arquivo XML que contém as declarações dos componentes utilizados na *Activity* principal "*MainActivity*". Nesse exemplo, o nome do arquivo é: "*activity_main*".

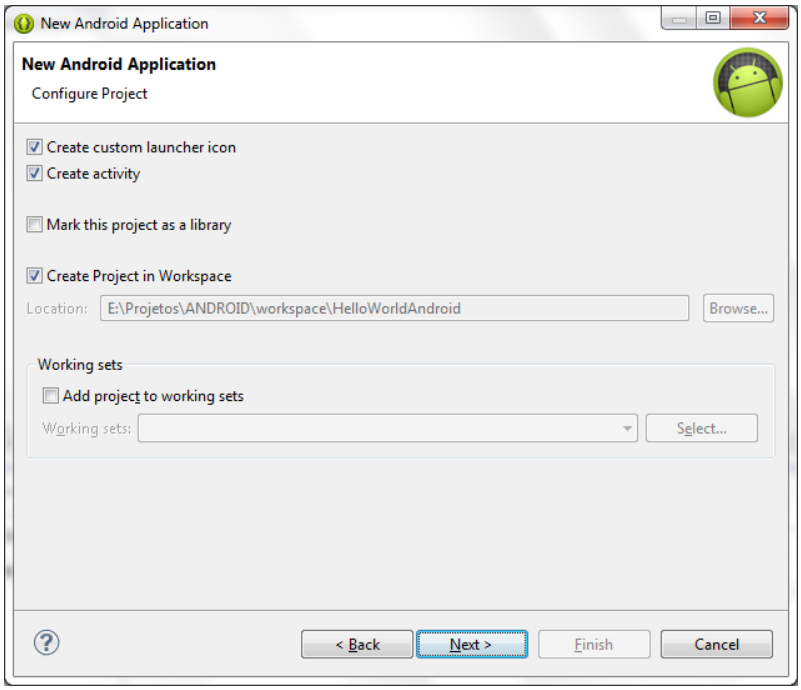

Fig. 17. Caixa de diálogo para atribuição do local onde o novo projeto Android será salvo.

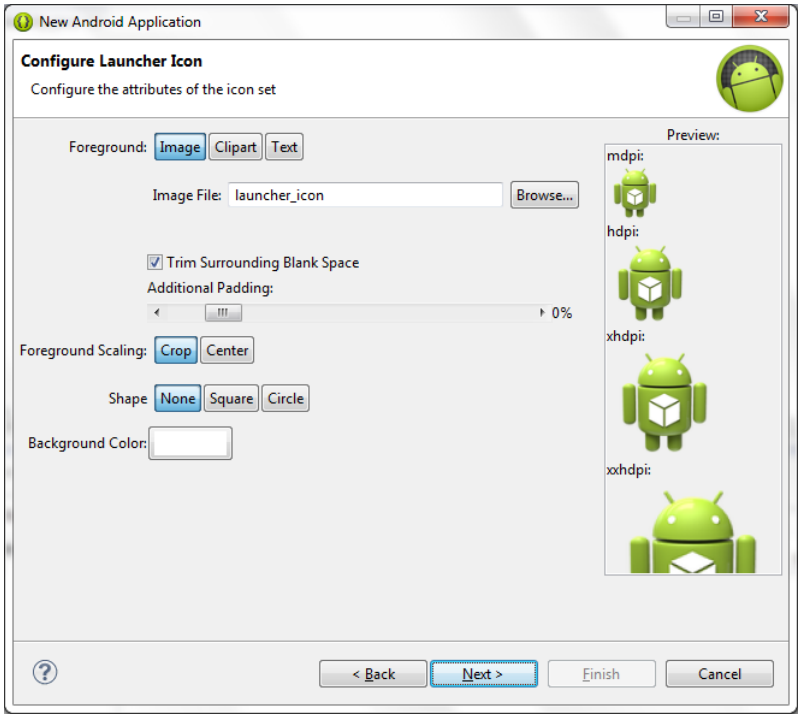

Fig. 18. Caixa de diálogo para atribuição do ícone do novo projeto Android.

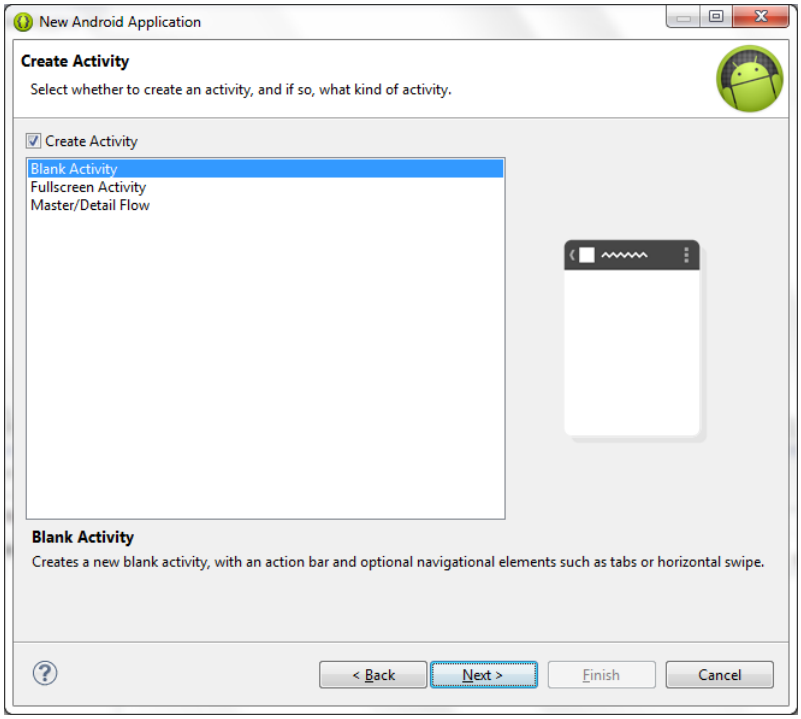

Fig. 19. Caixa de diálogo para atribuição do tipo da *Activity* do novo projeto Android.

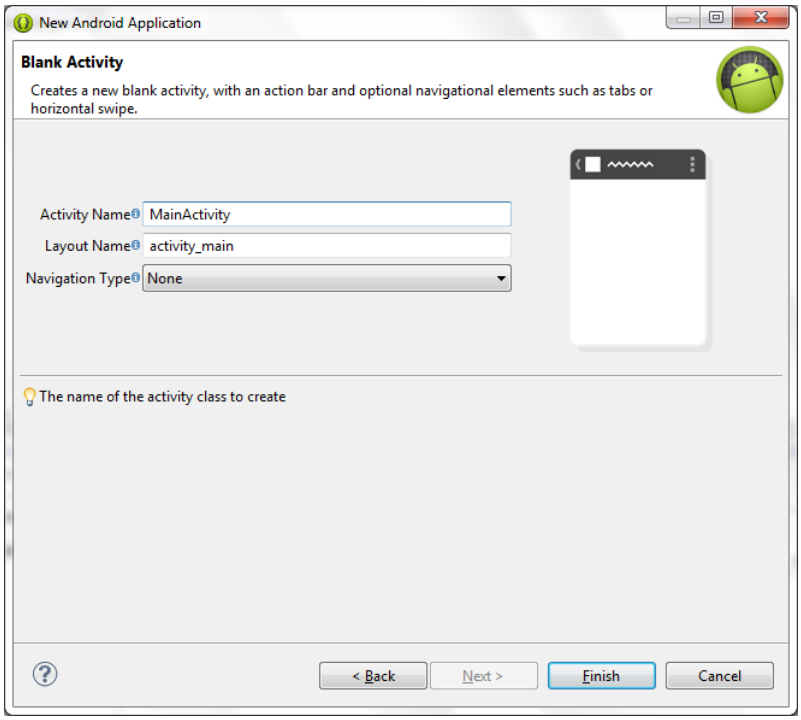

Fig. 20. Caixa de diálogo para atribuição do nome da *Activity* do novo projeto Android.

Após concluir as configurações na caixa de diálogo do novo projeto Android, o Eclipse apresenta através do painel Gerenciador de Pacote os arquivos do aplicativo. Nesse painel é exibida a hierarquia de pastas que o Eclipse criou automaticamente para o projeto. Clique nos ícones da seta ao lado da pasta "*HelloWorldAndroid*" e clique nos ícones da seta ao lado das pastas "*src*" e "*res*". Clique nas setas das pastas abertas ao lado de "*myapp.helloworld*", "*drawable*", "*layout*" e "*values*", então pode-se ver os arquivos Java e XML que Eclipse criou também. A **Fig. 21** mostra o painel Gerenciador de Pacote neste ponto, permitindo realizar uma inspeção dos arquivos do aplicativo.

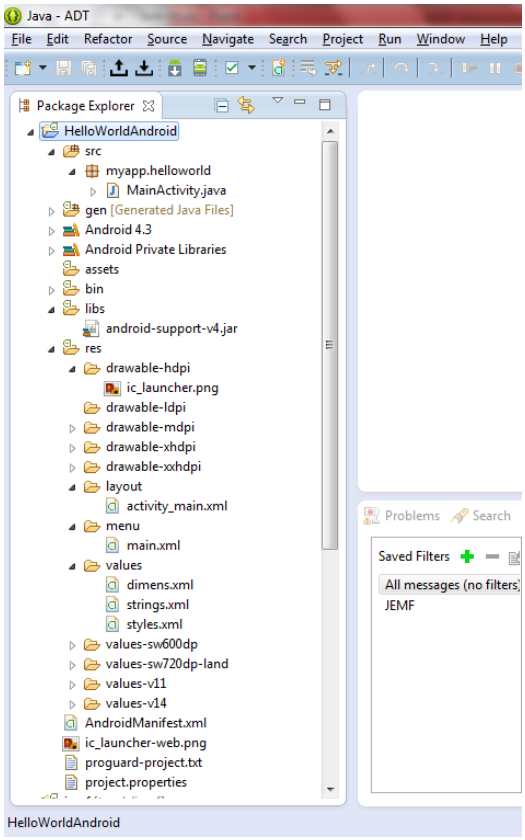

Fig. 21. Painel gerenciador de pacote do Eclipse.

O arquivo "*MainActivity.java*" contém a classe *Activity*, onde o Eclipse já definiu o código para criar a tela de interface do usuário para o aplicativo e definiu também o seu conteúdo (componentes) para essa interface através do arquivo "*activity_main.xml*".

4.7 Como importar o JEMF no novo Projeto de Aplicativo

Uma vez com o novo projeto criado, selecione o menu de propriedades para importar o arcabouço no projeto: "*Project* > *Properties*". Na caixa de diálogo "*Properties for HelloWorldAndroid*", selecione a opção "*Android*" na lista lateral esquerda para informar ao Eclipse qual o vínculo deseja ser criado, conforme mostrado na parte inferior da figura **Fig. 22**. Clique no botão "*Add*" do campo "*Library*" para continuar.

Na caixa de diálogo "*Project Selection*", selecione o arcabouço JEMF na lista de projetos e clique no botão "*OK*", conforme figura **Fig. 23**. Depois de selecionado, o arcabouço será apresentado na lista de referências do projeto de aplicativo (**Fig. 24**), clique no botão "*OK*" para finalizar a configuração. Portanto, o novo projeto estará apto a trabalhar com o arcabouço JEMF.

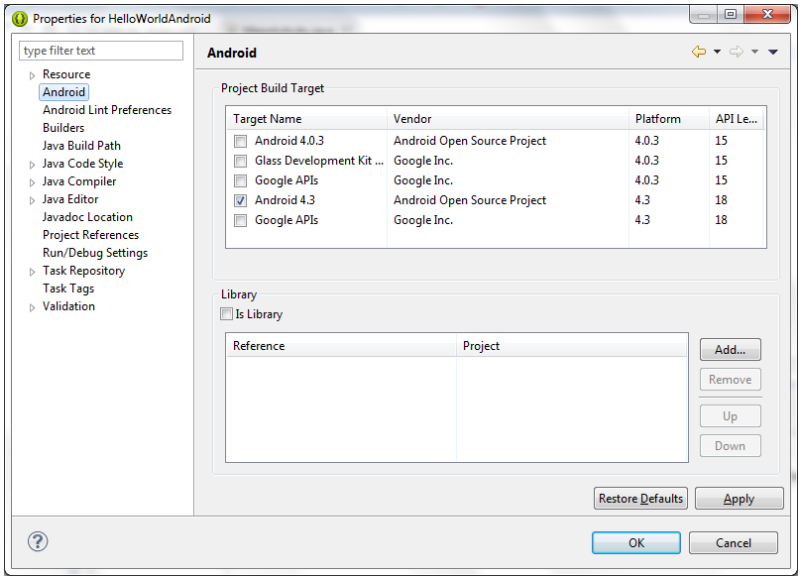

Fig. 22. Caixa de diálogo para importar o arcabouço JEMF no novo projeto Android.

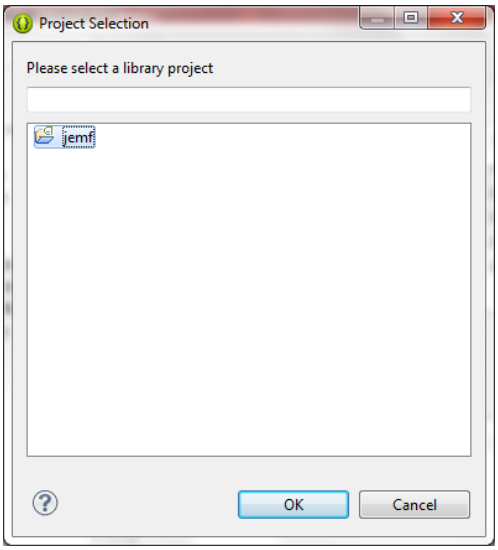

Fig. 23. Caixa de diálogo para importar o arcabouço JEMF no novo projeto Android.

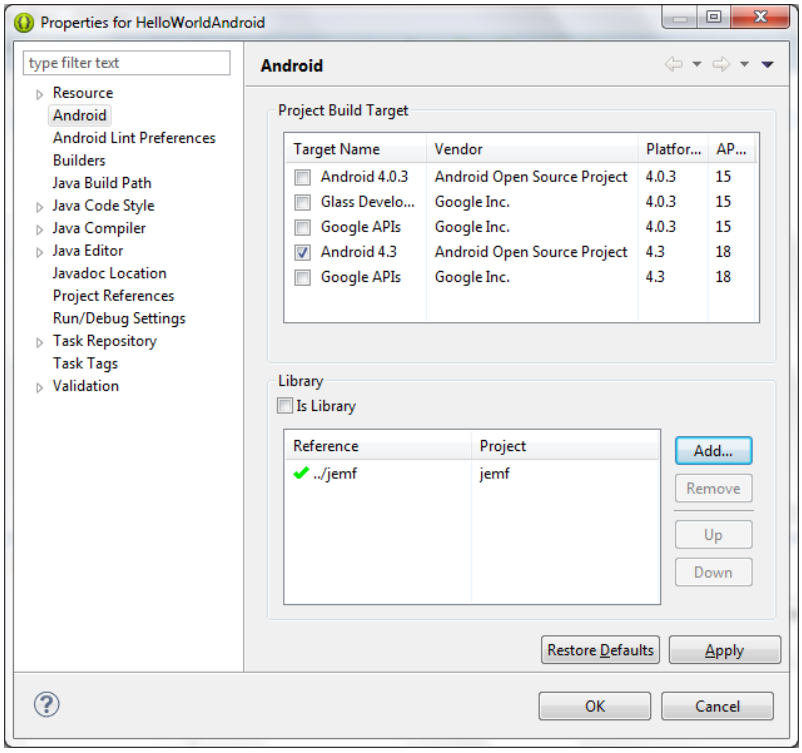

Fig. 24. Caixa de diálogo para importar o arcabouço JEMF no novo projeto Android.

4.8 Como usar o JEMF no novo Projeto de Aplicativo

Nesta seção é descrito como configurar um aplicativo para utilizar o arcabouço JEMF e como codificar algumas operações básicas através do arcabouço.

Configurando o Manifesto do Projeto de Aplicativo.

Este fragmento de código apresenta como é deve ser configurado o Manifesto do Projeto de Aplicativo (arquivo "*AndroidManifest.xml*") para que o novo aplicativo tenha acesso aos dados armazenados de todas as funcionalidades providas pelo arcabouço JEMF. Para explicar de maneira detalhada separamos o processo em quatro passos.

- A primeira configuração (**C1**) é a definição de uma permissão para acesso ao *ContentProvider* do aplicativo. O *ContentProvider* é uma das formas que o Android disponibiliza para um acesso controlado ao banco de dados. Para utilizar o JEMF é requerida esta configuração, caso contrário não será possível realizar os métodos de inclusão, alteração ou exclusão de dados pelo JEMF e que ocasionará exceções no aplicativo.
- A segunda configuração (**C2**) é a definição de uma permissão que habilitará o aplicativo salvar arquivos em um diretório externo, diferente do diretório da aplicativo. Esta configuração é utilizada pelo JEMF para salvar arquivos de log que poderão ser gerados durante sua execução.
- A terceira configuração (**C3**) é a definição do *ContentProvider*. No *ContentProvider* são definidos os atributos como nome da classe do JEMF que estão programados os métodos de acesso aos dados, a autoridade que é uma assinatura única para identificar o *Con-*

tentProvider, a permissão que foi definida no passo anterior (C1) e se o *ContentProvider* estará habilitado.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
[...]

  <!– [C1] -->
  <permission
android:name="br.ufrj.ppgi.jemf.mobile.provider.ACCESS_DATA"
android:protectionLevel="signature"
  />

  <!– [C2] -->
  <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"
  />

  <application
    [...]
    <activity
      [...]
    </activity>

  <!– [C3] -->
  <provider
  android:name="br.ufrj.ppgi.jemf.mobile.provider.CustomProvider"
  android:authorities="br.ufrj.ppgi.jemf.mobile.provider"
  android:label="@string/app_name"
  android:enabled="true"
  android:exported="true"
  android:permission="br.ufrj.ppgi.jemf.mobile.provider.ACCESS_DATA"
  >
  </provider>

  [...]
  </application>
</manifest>
```

Consultando dados no Banco de Dados e carregando-os no componente *ListView*.

Este fragmento de código apresenta como é possível carregar dados armazenados no banco de dados para um componente *ListView* de forma assíncrona através do arcabouço JEMF. Para explicar de maneira detalhada separamos o processo em seis passos.

- O ponto de partida para instanciar o JEMF é através da classe *EmergencyManagementFactory*. Ao se criar uma instância dessa classe (**C1**), temos acesso aos métodos necessários para integrar JEMF com o aplicativo e usar operações para carregar e exibir os dados armazenados.
- O componente *ListView* é uma das possiblidades disponíveis pela API do Android para interação e exibição de dados armazenados no dispositivo. Este componente é geralmente o mais utilizado para realizar esta operação (**C2**), visto que ele apresenta os dados em uma lista de elementos.
- O componente *CursorAdapter* é responsável pelo armazenamento temporário de um conjunto de dados na memória virtual do dispositivo móvel. Portanto, através dele é possível navegar e manipular os valores registrados, permitindo a exibição desses dados no

formato desejado. O arcabouço JEMF disponibiliza um método (**C3**) para a criação do objeto *CursorAdapter* de acordo com uma das funcionalidades já apresentadas. Neste exemplo é requisitado um objeto da funcionalidade que gerencia as Emergências (*EnumFeatureType.EMERGENCY*). Assim, ao ser instanciado o objeto *CursorAdapter*, ele já estará configurado e formatado para a exibição dos dados de Emergências.

- O componente *ListView* oferece uma interface para anexação de um objeto *CursorAdapter*. Após instanciar o objeto *CusorAdapter*, podemos então anexá-lo com o componente *ListView* (**C4**) e quando os dados de Emergências forem carregados do banco de dados, ele passará a exibi-los como uma lista vertical.
- Até o passo anterior, configuramos os componentes que formatam e exibem os dados, porém o aplicativo ainda não foi configurado para acessar e ler esses dados do banco de dados. Para realizar essa operação (**C5**), utilizaremos o componente *Loader* que é responsável por carregar os dados de forma assíncrona, melhorando o desempenho do aplicativo. Logo, é necessária a criação de um objeto *LoaderCallbacks* que contém a consulta que será executada no banco de dados. O arcabouço JEMF disponibiliza um método para instanciar esse objeto de maneira semelhante ao *CursorAdapter*.
- Por último, precisamos iniciar o *Loader* (**C6**) que carregará os dados que serão exibidos apropriadamente no *ListView*.

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin" >

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:text="Add emergency" />

    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/button"
        android:layout_centerHorizontal="true" >
    </ListView>

</RelativeLayout>
```

```
import br.ufrj.ppgi.jemf.mobile.FeatureHolder.EnumFeatureType;
import br.ufrj.ppgi.jemf.mobile.factory.EmergencyManagementFactory;
import android.app.LoaderManager.LoaderCallbacks;
import android.database.Cursor;
import android.widget.CursorAdapter;
import android.widget.ListView;

public class MainActivity extends Activity {

// Loader ID.
private final int loaderID = 1;

@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);

// [C1] Instantiate JEMF
  EmergencyManagementFactory emf =
EmergencyManagementFactory.newInstance(this);

// [C2] Get the ListView
  ListView listView = (ListView) findViewById(R.id.listView);

// [C3] Instantiate a Custom CursorAdapter to set in ListView
  CursorAdapter cursorAdapter =
emf.getMainFactory(EnumFeatureType.EMERGENCY).getAdapter();

// [C4] Set CustomAdapter to ListView
  listView.setAdapter(cursorAdapter);

// [C5] Instantiate a Custom LoaderCallbacks to read data from Data-
base
  LoaderCallbacks<Cursor> loaderCall =
emf.getMainFactory(EnumFeatureType.EMERGENCY).getLoader(null, null,
null);

// [C6] Load content by Custom LoaderCallbacks and LoadManager
  getLoaderManager().initLoader(loaderID, null, loaderCall);
}

[...]
}
```

Inserindo ou Editando dados no Banco de Dados.

Este fragmento de código apresenta como é possível incluir ou alterar dados e armazená-los no banco de dados através do arcabouço JEMF. Para explicar de maneira detalhada o processo é separado em quatro passos.

- Neste exemplo continuaremos com a funcionalidade que gerencia as Emergências. O JEMF disponibiliza um conjunto de classes concretas que representam as entidades que envolvem a resposta a emergências, já apresentadas em sessão anterior. Essas classes modelos são fundamentais para a manipulação dos dados no aplicativo. Com isso, começamos esta operação de inclusão ou alteração (**C1**) com a instanciação de um objeto *Emergency*.
- Em seguida, informamos valores para cada atributo que compõe a Emergência (**C2**) através da interface disponibilizada pelo objeto (Métodos *Get*() e *Set*()).
- Ao instanciar o JEMF através da classe *EmergencyManagementFactory*, (**C3**) temos acesso aos métodos necessários para incluir ou alterar os dados no banco de dados.
- O JEMF disponibiliza um método para salvar o objeto *Emergency* de acordo com a funcionalidade que gerencia as Emergências (*EnumFeatureType.EMERGENCY*). Assim, precisamos passar o objeto *Emergency* para ser executado pelo método de forma adequada (**C4**). Caso dos dados sejam válidos, o JEMF gravará a nova Emergência ou modificará a Emergência no banco de dados.

```
public class MainActivity extends Activity {
[...]

@Override
protected void onClick(View v) {

// [C1] Instantiate the Emergency
  Emergency emergency = new Emergency();

// [C2] Set Emergency data
  emergency.setName("Emergency A");
  emergency.setActivated(true);
  emergency.setLevel(EnumLevel.MUNICIPAL);
  emergency.setStartDate(new Date());
  emergency.setEndDate(new Date());
  emergency.setType(EnumEmergencyType.FIRE);

// [C3] Instantiate JEMF
  EmergencyManagementFactory emf =
EmergencyManagementFactory.newInstance(this);

// [C4] Save the Emergency
  emf.getMainFactory(EnumFeatureType.EMERGENCY).getManager().
save(emergency);
}

[...]
}
```

Excluindo dados no Banco de Dados.

Este fragmento de código apresenta como é possível excluir dados armazenados no banco de dados através do arcabouço JEMF. Para explicar de maneira detalhada separamos o processo em quatro passos.

- Neste exemplo continuaremos com a funcionalidade que gerencia as Emergências. Antes de realizarmos a exclusão de uma Emergência, o exemplo (**C1**) demostra que precisamos capturá-la dentro a lista de Emergências exibida pelo componente *ListView*. O *ListView* permite através de um método capturar o elemento selecionado pelo usuário do aplicativo. De posse do elemento selecionado, convertemos para o objeto *Cursor*, contendo os dados da Emergência em forma de tupla.
- Em seguida (**C2**), precisamos validar se o *Cursor* é um objeto válido para prevenir que exceções ocorram no aplicativo. Caso positivo, prosseguimos com a operação.
- Ao instanciar o JEMF através da classe *EmergencyManagementFactory*, (**C3**) temos acesso aos método necessário para excluir os dados no banco de dados.
- O JEMF disponibiliza um método para deserializar o objeto *Cursor* (**C4**) e convertê-lo em um objeto *Emergency*, necessário para ser executar a método de exclusão de forma adequada.
- Por fim, através do JEMF executamos o método que exclui a Emergência (**C5**) selecionada de acordo com a funcionalidade que gerencia as Emergências (*EnumFeatureType.EMERGENCY*).

```
public class MainActivity extends Activity {
[...]

@Override
protected void onClick(View v) {

// [C1] Get the Emergency from the ListView
  Cursor cursor = (Cursor) listView.getAdapter().
getItem(listView.getSelectedItemPosition());

// [C2] Check if Cursor is valid
 if (cursor != null) {

// [C3] Instantiate JEMF
  EmergencyManagementFactory emf =
EmergencyManagementFactory.newInstance(this);

// [C4] Convert Cursor into Emergency
  Emergency emergency = (Emergency)
emf.getMainFactory(EnumFeatureType.EMERGENCY).getManager().
deserialize(cursor);

// [C5] Delete the Emergency
  emf.getMainFactory(EnumFeatureType.EMERGENCY).getManager().
remove(emergency);
 }
}

[...]
}
```

4.9 Executando o Projeto de Aplicativo

Para compilar e executar o aplicativo, clique com botão direito do mouse sobre o ícone da pasta "*HelloWorldAndroid*" no Gerenciador de Pacote do Eclipse e selecione "*Run As > Android Application*". O Eclipse irá compilar o aplicativo e, em seguida, abrirá uma janela do emulador para exibir um telefone virtual na qual ele será executado. Quando o emulador é iniciado, ele exibirá a tela do padrão do smartphone, simulando uma imagem de fundo e ícones do sistema Android, e permitindo navegar nos aplicativos instalados.

Para executar o aplicativo no emulador, é preciso clicar no botão de Menu na área média inferior da tela, ou use a tecla Home para exibir os ícones de aplicativos e em seguida, selecione o ícone do aplicativo "*HelloWorldAndroid*" para executá-lo, ou seja, use a interface de telefone virtual para encontrar e executar o aplicativo como se fosse na telefone real. Então, o aplicativo não só funcionará no emulador, mas poderá ser instalado em qualquer smartphone com uma versão do Android que seja compatível ao mesmo (Android versão 4.0.3 ou superior).

5 Licença

JEMF é um projeto de código aberto e licenciado com a Licença Pública Geral (*GNU General Public License*). A GNU GPL ou simplesmente GPL, é a designação da licença para software livre idealizada em 1989, no âmbito do projeto GNU da Free Software Foundation (FSF) [20]. A licença GPL foi originalmente publicada em Janeiro de 1989. No entanto, este projeto é baseado na **GPL Versão 3** (GPLv3), onde seu primeiro esboço foi publicado em 16 de Janeiro de 2006, sendo a versão final lançada em 29 de Junho de 2007.

Em termos gerais, a GPL baseia-se em quatro liberdades:

1. A liberdade de executar o programa, para qualquer propósito (liberdade nº 0).
2. A liberdade de estudar como o programa funciona e adaptá-lo para as suas necessidades (liberdade nº 1). O acesso ao código-fonte é um pré-requisito para esta liberdade.
3. A liberdade de redistribuir cópias de modo que você possa ajudar ao seu próximo (liberdade nº 2).
4. A liberdade de aperfeiçoar o programa, e liberar os seus aperfeiçoamentos, de modo que toda a comunidade se beneficie deles (liberdade nº 3). O acesso ao código-fonte é um pré-requisito para esta liberdade.

Com a garantia destas liberdades, a GPL permite que os programas sejam distribuídos e reaproveitados, mantendo, porém, os direitos do autor por forma a não permitir que essa informação seja usada de uma maneira que limite as liberdades originais. A licença não permite, por exemplo, que o código seja apoderado por outra pessoa, ou que sejam impostos sobre ele restrições que impeçam que seja distribuído da mesma maneira que foi adquirido.

A GPL está redigida em inglês e atualmente nenhuma tradução é aceita como válida com o argumento de que há o risco de introdução de erros de tradução que poderiam deturpar o sentido da licença. Deste modo, neste projeto se manteve a obrigatoriedade de distribuir o texto oficial em inglês com o código do arcabouço.

Referências

1. Angermann, M., Khider, M., Frassl, M., Lichtenstern, M.: DMT - An integrated disaster management tool. 1st International Conference on Disaster Management and Human Health Risk, United Kingdom (2009).
2. Araújo, F.C.S.: A conceptual framework for systems development on mobile devices to support emergency management. Master's Thesis, Federal University of Rio de Janeiro, Brazil (2012)
3. Ellis, C.A., Gibbs, S.J., Rein, G.L.: Groupware - Some Issues and Experiences. Communications of the ACM 34(1) (1991) 38–58
4. Fayad, M., Schmidt, D.C.: Object-oriented Application Frameworks. Communications of the ACM 40(10) (1997) 32–38
5. Humayoun, S., Catarci, T., de Leoni, M., Marrella, A., Mecella, M., Bortenschlager, M., Steinmann, R.: Designing Mobile Systems in Highly Dynamic Scenarios: The WORKPAD Methodology. Knowledge, Technology & Policy, Springer Netherlands 22(1) (2009) 25-43
6. Ibarra, M., Monares, A., Ochoa, S.F., Pino, J.A., Suarez, D.: A mobile collaborative application to reduce the radio traffic in urban emergencies. IEEE 16th International Conference Computer Supported Cooperative Work in Design (2012) 358–365
7. Jackson, A. W.: Android Apps for Absolute Beginners. Apress, (2011) 115 - 145
8. Johnson, R., Foote, B.: Designing Reusable classes. Journal of Object-Oriented Programming (JOOP) 1(2) (1988) 22–35
9. Joshi, S.G.: Exploring map-based interfaces for mobile solutions in emergency work. Master's Thesis, University of Oslo, Norway (2011)
10. Kanchanasut, K., Tunpan, A., Awal, M.A., Das, D.K., Wongsaardsakul, T., Tsuchimoto, Y.: DUMBONET: A multimedia communication system for collaborative emergency response operations in disaster-affected areas. International Journal of Emergency Management 4(4) (2007) 670-681
11. Kienzle, J., Guelfi, N., Mustafiz, S.: Crisis Management Systems: A Case Study for Aspect-Oriented Modeling. Aspect-Oriented Software Development VII, Berlin, Heidelberg (2010) 1-22
12. Luyten, K., Winters, F., Coninx, K., Naudts, D., Moerman, I.: A situation-aware mobile system to support fire brigades in emergency situations. OTM 2006 workshops 2(A) (2006) 1966-1975
13. Meissner, A., Wang, Z., Putz, W., Grimmer, J.: MIKoBOS - A Mobile Information and Communication System for Emergency Response. 3rd International Conference on Information Systems for Crisis Response and Management (2006) 978–990
14. Padilha, R.P.: Collaboration between command and operations teams during emergency response: a proposal using mobile computing. Master's Thesis, Federal University of Rio de Janeiro, Brazil (2010)
15. Pimentel, M., Fuks, H.: Collaborative Systems. Elsevier, Rio de Janeiro, Brazil (2011)
16. Ponserre, S., Guha-Sapir, D., Vos, F., Below, R.: Annual disaster statistical review 2011: the numbers and trends. Université Catholique de Louvain, Brussels, Belgic (2012)
17. Roberts D., Johnson R.: Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks. 3rd Conference on Pattern Languages and Programming (1996)
18. Schächinger, U., Kretschmer, R., Röckelein, W., Neumann, C., Maghsudi, M., Nerlich, M.: NOAH – A Mobile Emergency Care System. European Journal of Medical Research 2000(5) (2000) 13–18
19. Silva, A.J.D.: A mobile computing system to support first responder in radiological emergency. Master's Thesis, Federal University of Rio de Janeiro, Brazil (2012)
20. GNU General Public License, http://pt.wikipedia.org/wiki/GNU_General_Public_License.
21. ADT Bundle, http://developer.android.com/tools/help/adt.html
22. API Level, http://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels
23. Android Plataforms, http://developer.android.com/about/dashboards/index.html#platform
24. SVN, http://subversion.apache.org/
25. Subversive, http://www.eclipse.org/subversive/
26. Android Activity, http://developer.android.com/reference/android/app/Activity.html

27. Android Fragment, http://developer.android.com/guide/components/fragments.html
28. Android Fragment inside Activity, http://airpair-blog.s3.amazonaws.com/wp-content/uploads/2014/02/fragments.png
29. Android CursorAdapter, http://developer.android.com/reference/android/widget/CursorAdapter.html
30. Android Cursor, http://developer.android.com/reference/android/database/Cursor.html
31. Android ListView, http://developer.android.com/reference/android/widget/ListView.html
32. Android Loader, http://developer.android.com/reference/android/content/Loader.html
33. Android ContentProvider, http://developer.android.com/reference/android/content/ContentProvider.html
34. Android SQLiteOpenHelper , http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html\
35. JavaBean, http://pt.wikipedia.org/wiki/JavaBeans