Sobre o sistema

Explicação formal:

As decisões tomadas pelo comandante da operação em situações de emergência devem ser feitas rapidamente para salvar vidas. Para evitar decisões tardias ou ruins, o comandante deve construir uma consciência situacional. O fluxo de chegada irregular de informação, a incerteza a sobrecarga de informações e a falta de persistência são os principais fatores que dificultam essa tarefa.

Para minimizar esses efeitos, propomos uma arquitetura composta de dispositivos móveis e um sistema de apoio à decisão para ser usado no posto de comando. O ponto principal no projeto do sistema é a sobrecarga cognitiva. Portanto, heurísticas sobre o uso das informações por comandantes experientes foram reunidas e implementadas.

Tanto decisões atrasadas quanto imprecisas podem implicar mais danos às vítimas, para a economia ou mesmo pôr em risco a equipe de resposta. Nestes casos, é vital que essas pessoas tenham informações tanto atualizadas quanto precisas, para que possam compreender a situação, de forma a fazer as suas avaliações e decisões o mais seguro e correto possível.

- Para aumentar a rapidez de acesso à informação e evitar a sobrecarga dessa, somente os dados mais importantes são exibidos em um primeiro momento. Porém, a descrição completa desses também está disponível para acesso.

Comunicação

- Dispositivos se conectam à rede de internet, e a partir de seus números de IP, ligam-se via Sockets em torno de um “dispositivo-servidor”, a partir da interface disponibilizada pelo sistema

Conclusão

- O estudo do tratamento de emergências gerou requisitos para sistemas de informação que apoiem estas situações (requisitos de informação, arquitetura, mobilidade, etc.)
- Nova maneira de disponibilizar a informação (priorizações); além de acelerar o entendimento sobre um ponto de emergência utilizando, para isso, os relacionamentos com outros pontos.

Sobre o sistema

Explicação informal:

Estamos desenvolvendo um sistema que será usado como protótipo para futuras implementações, cujo objetivo é auxiliar na tomada de decisões dos comandantes das equipes de emergência. A ideia principal do projeto é que haja um dispositivo central (um tablet, idealmente) que sirva de central de troca de dados entre todos os dispositivos-clientes que estarão junto com as equipes de resposta em campo.

Inicialmente, a ideia do prof. era que tivéssemos uma rede ad-hoc entre os dispositivos, de forma que não fosse necessária nenhuma conexão wi-fi disponível. Porém, foi concluido após algumas pesquisas de que esse tipo de rede é muito instável e difícil de ser implementada com android, uma vez que podem existir muitos dispositivos diferentes rodando a aplicação, e nem todos sempre responderão da mesma forma à conexão ad-hoc.

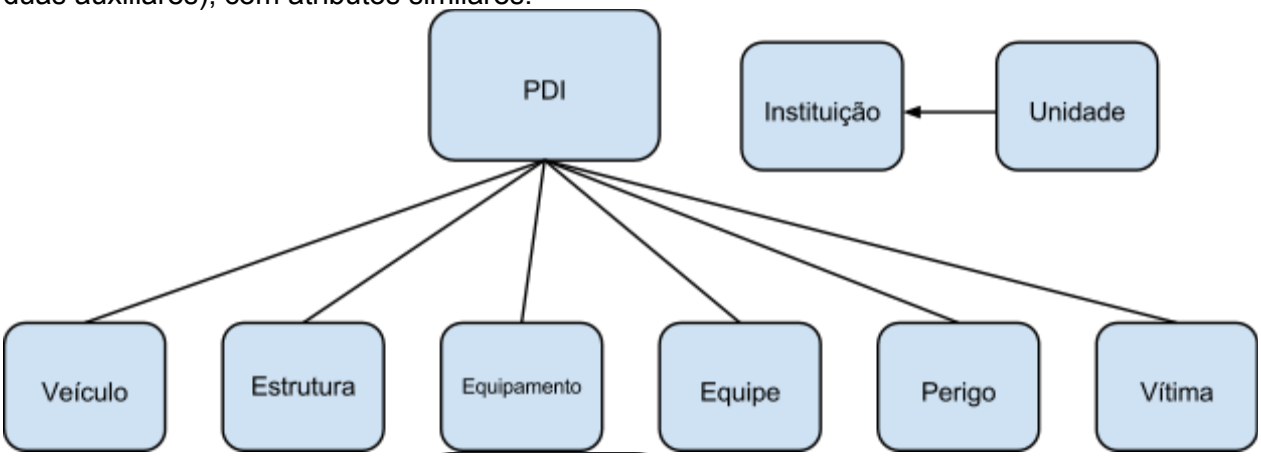
O código já existente, em teoria, deveria servir somente para o dispositivo-servidor, e portanto, a versão dos dispositivos-clientes nada mais seria do que a aplicação original, com algumas funcionalidades que não seriam úteis em campo removidas (como por exemplo, visualização de detalhes de pontos-de-interesse em outras áreas distantes daquela onde se encontra a equipe de campo).

Obs: Como uma aplicação android, existe a possibilidade de haver atualizações do aplicativo. Porém, evidentemente, não podemos usar os métodos comuns de atualização, através do Google Play, uma vez que, se disponibilizado ali, a aplicação ficaria pública para qualquer um baixar - obviamente, não queremos que esta aplicação seja disponibilizada para o grande público. Com isso, durante o desenvolvimento, utilizei um método “gambiarra” para fazer as atualizações em todos os dispositivos onde testei (tanto meus devices, quanto os devices do GRECO - tablets e telefones), que passava por arquivos na minha própria pasta do Dropbox. Obviamente, isso precisará ser alterado já no próximo build *(mais detalhes abaixo)*.

Modelagem

A modelagem da aplicação foi feita de acordo com o material que me foi entregue como resultado da pesquisa de mestrado feita pelo André Engelbrecht (nome do artigo: **A Decision Support System for Medium-Sized Emergencies**). Ele entrevistou profissionais da área de emergências, tanto no Brasil quanto em outros países da América Latina, e especificou que tipo de entidades são as mais comuns no contexto do tratamento de emergência (tudo isto consta nos artigos que foram enviados por email, e nas folhas que estavam junto ao tablet).

Foram definidas oito entidades (seis delas sendo pontos de interesse - PDI, e outras duas auxiliares), com atributos similares:



Além disso, foram especificados relacionamentos entre estas entidades, que foram implementadas no sistema, e podem ser resumidas na seguinte tabela (retirada do artigo):

Entity	Related to
Risk	Team, Victim
Structure	Risk, Team, Victim
Victim	Risk, Team, Structure
Team	Risk, Structure, Equipment
Vehicle	Team, Equipment
Equipment	Team, Vehicle

Cada uma das entidades tem um grupo específico de atributos, para maior detalhamento dos dados da aplicação. Alguns destes atributos tem maior prioridade de exibição, enquanto outros são secundários, mas ainda assim estão disponíveis em um segundo momento para o usuário. Cada um destes atributos de cada entidade está presente na tabela que enviei por email.

Código

O código da aplicação está dividido atualmente em quatro pacotes:

1. **br.com.site**: Activities, Broadcast Receivers, e algumas classes de biblioteca. Procurei manter este pacote como *controlador* da aplicação.
2. **br.com.site.model**: Classes do modelo
3. **br.com.site.view**: Classes auxiliares de instanciação de views personalizadas, como por exemplo os balões do mapa, marcadores, paineis flutuantes, etc.
4. **br.com.site.web**: Classes de comunicação com a web, e com outros dispositivos (tanto as classes de servidor e cliente constam na aplicação. Em tempo de execução, os devices podem ser configurados como servidor/cliente, e a respectiva classe será ativada).

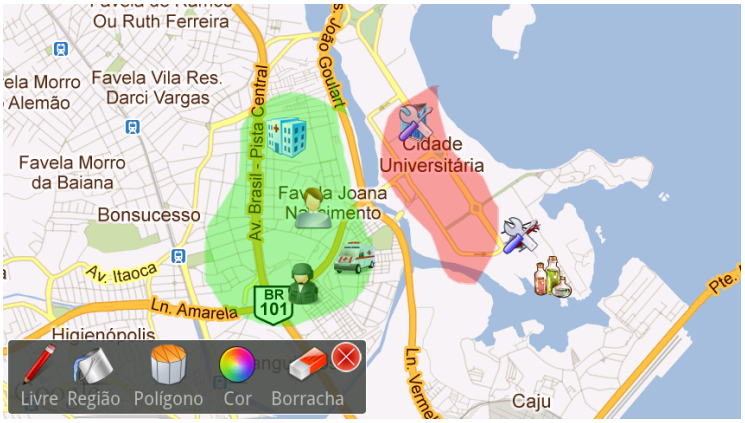
br.com.site

Pacote contendo as activities do sistema, incluindo a activity principal (MainActivity), que exibe o mapa, e se relaciona com todas as outras classes do sistema, e as activities de criação de PDIs (extendendo a classe NovoPDI), além da classe de escolha de exibição dos PDIs (seguindo a ideia do artigo de só exibir o que for agregar às decisões do usuário), e de configuração de conexão do servidor com clientes.

O pacote também contém o Broadcast Receiver que “liga e desliga” um flag para sabermos se o dispositivo está online.

Também está incluso um parser de XML para podermos ler os PDIs que foram transmitidos via rede, ou salvos na memória do dispositivo para acesso posterior (algo como uma função “carregar emergência”, que será descrita mais abaixo).

Além disso, algumas classes auxiliares para o mapa, como a classe que trata a forma como são feitos desenhos no mapa (de regiões ou linhas), que trata a interação com os balões, e com os menus flutuantes.



br.com.site.model

Classes de modelo, seguindo a especificação do André, seus atributos com boilerplates, etc.

As classes que herdam de PDI tem um método getXML que serve para “serializar” os objetos para serem enviados via web para outros dispositivos. A classe PDI gerará o arquivo XML com as tags e dados a partir de um mapa de relacionamentos em cada classe-filha.

br.com.site.web

A classe MessageListener servirá como servidor da aplicação, conectando-se via Socket com os clientes através do seu IP, e poderá receber mensagens do tipo criar PDI, editá-los, excluí-los, etc. O fluxo de troca de mensagens pode ocorrer de duas formas:

a. O próprio dispositivo servidor alterou algum PDI
    i. O servidor deverá enviar uma mensagem para todos seus clientes conectados para que os mesmos alterem o determinado PDI da mesma forma
b. Um cliente alterou algum PDI
    i. O cliente enviará ao servidor uma mensagem para que ele altere aquele PDI
    ii. O servidor ficará responsável por retransmitir a mensagem para todos os outros clientes conectados, de forma que seja mantida a consistência das informações entre todos os dispositivos

Obs.: Note que, após cada envio de mensagem, o dispositivo “enviante” fica esperando uma resposta do “receptor” de que a mensagem foi recebida com sucesso. Para isso, o “receptor” responderá com uma mensagem do tipo ACK (acknowledge) ou NACK (not acknowledge).

Já a classe ClientThread nada mais é do que uma linha de execução que fica em loop aguardando por alguma mensagem que possa chegar a qualquer momento. Tanto dispositivos servidor quanto clientes podem executar esta linha de execução.

*Obs*².: A classe DownloadApk, como o nome diz, servirá para baixar uma possível atualização do aplicativo (como referenciei acima, numa url no meu DropBox - coisa que deve ser alterada), salvá-la no cartão SD do aparelho, e iniciar o *Intent* de instalação do arquivo.

O que falta

Como eu já havia comentado, ainda é preciso

a. Limpar um bocado de código, algumas gambiarras e más práticas de programação, que foram implementadas enquanto eu ainda aprendia a usar o framework android
b. Implementar um banco de dados (tanto local, no próprio dispositivo, quanto externo, para servir ao servidor e outros clientes, mantendo-os sempre consistentes e atualizados), de forma que os dispositivos comuniquem-se com ele antes de enviar os dados para outros conectados na rede.
c. A partir destes bancos de dados (local e externo), implementar uma forma do usuário poder tratar várias emergências com um mesmo dispositivo - isto é, poder carregar dados de PDIs, desenhos, marcações, etc, mesmo após já ter fechado o app alguma vez (para casos em que o tratamento das emergências dure mais de um dia, por exemplo), que podem ser carregados tanto da memória interna do dispositivo, quanto do banco de dados online do sistema.
d. Com a chegada dos novos tablets com maior resolução (xhdpi), pode ser uma boa ideia alterar os layouts que existem atualmente para que dêem suporte à essas telas, provavelmente usando a nova API de fragments.
e. A conectividade entre os dispositivos (por questão de simplicidade para os testes, etc) é feita atualmente diretamente via IP. Isto é, para conectar-me a um dispositivo usando o sistema, eu precisaria saber o IP do mesmo. Seria muito interessante encontrar uma forma de implementar para o sistema uma forma de nomear estes IPs, usando um sistema DNS, ou então, fazer este tratamento através do computador onde estará o banco de dados central do sistema (o problema desta implementação seria o caso onde este computador estivesse offline - dessa forma, nenhum dispositivo conseguiria ser encontrado por outros, por isso, encorajo que encontre uma forma utilizando o DNS).
f. Por fim, novamente por questão de simplicidade, eu hospedei alguns arquivos do sistema na minha pasta do DropBox e na minha página pessoal do DCC. Uma vez que você consiga um computador que sirva como servidor do sistema, você poderá mover estes arquivos das minhas pastas para lá, e assim, alterar estas URLs no código do sistema. Vou manter os arquivos online por enquanto, até que você consiga outro local para armazená-los, mas é interessante que você pelo menos mova-os para suas pastas pessoais no caso de precisar alterá-los de alguma forma, uma vez que você não tem acesso às minhas pastas.