2
Aplicações Sensíveis ao Contexto

O desenvolvimento de aplicações sensíveis ao contexto tem adotado uma abordagem que assume o uso de *middlewares* capazes de coletar, gerenciar e disseminar informações de contexto, para assim reduzir a complexidade exigida no desenvolvimento dessas aplicações (Henricksen & Indulska, 2005). De fato, atualmente as ASCs assumem o papel de consumidores das informações providas por *middlewares* de gerenciamento de contexto e são desenvolvidas a partir das APIs disponibilizadas por tais *middlewares*. Desta forma, em nossos estudos utilizamos aplicações baseadas em APIs de *middlewares* e, particularmente, optamos por usar MoCA (Rubinsztejn et al, 2004) devido a duas razões principais: documentação disponível e suporte ao programador.

Neste capítulo, são apresentadas as definições básicas de ASCs (Seção 2.1) e, a fim de se efetuar a análise de requisitos comuns em ASCs (Seção 2.3), são também apresentados os serviços de MoCA (Seção 2.1.2) e duas ASCs desenvolvidas a partir de suas APIs (Seção 2.2). O levantamento dos requisitos de ASCs será considerado na elaboração do *framework* CAAF (Capítulo 4) para reuso de conceitos especificamente relacionados à sensibilidade ao contexto que podem ser convenientemente tratados a partir de ESSMA (Capítulo 3).

2.1.
Definições Básicas de ASCs

Contexto é o elemento básico manipulado em ASCs. Entretanto, apesar de fundamental em ASCs, a definição de contexto tem evoluído muito nos últimos anos e isto tem ocorrido paralalemente ao desenvolvimento da área de pesquisa de Computação Ubíqua (Weiser, 1991) e da disponibilidade de serviços providos por *middlewares* para desenvolvimento de ASCs (Rubinsztejn et al, 2004). A seguir, é apresentada a definição de contexto utilizada neste trabalho, além de uma visão geral sobre MoCA, o *middleware* para desenvolvimento de ASCs utilizado em nossos estudos de casos.

2.1.1. Contexto e ASCs

Active Maps (Schilit & Theimer, 1994) foi o primeiro trabalho a utilizar explicitamente o conceito de sensibilidade ao contexto. Informações como localização, identificação de pessoas e objetos, bem como *mudanças que ocorrem sobre estes objetos* ao longo do tempo, foram utilizadas como contextos. As ASCs foram definidas então como aplicações informadas sobre contextos ou que se adaptam a um contexto (Schilit & Theimer, 1994).

Mais tarde, Brown (1996) utilizou outros exemplos de informações de contexto em sua arquitetura *Stick-e Note*: localização, presença de objetos e pessoas, temperatura e, de forma mais geral, *qualquer fator ambiental* possível de ser percebido e que pode exercer influência sobre as atividades de um sistema computacional. De acordo com Brown, Hull et al (1997) e Pascoe (1998) evoluíram a definição de ASCs. Segundo estes autores, uma aplicação é sensível ao contexto quando possui a habilidade de detectar, interpretar e responder a estímulos provenientes do ambiente, do usuário e dos dispositivos de computação.

Posteriormente, Schmidt et al (1998) propuseram uma categorização geral para contextos: fatores humanos e ambiente físico. A categoria de fatores humanos é também subdividida em *usuário* (hábitos, estado mental ou fisiológico), *ambiente social* (proximidade de outros, relacionamento social e tarefas colaborativas) e *tarefas* (objetivos de atividades ou objetivos gerais do usuário). A categoria ambiente físico possui as subcategorias de *localização* (coordenada GPS ou relativa a um contexto), infra-estrutura (a computação ao redor e interações no ambiente) e *condições* (condições físicas típicas do ambiente, como ruído de fundo, nível de luz do ambiente, brilho, etc.).

Em seguida, Dix et al (1999) definiram uma taxonomia para contexto em termos das *interações humano-computador*, onde os autores argumentam que um dispositivo móvel opera em um *contexto bastante amplo*, incluindo a infra-estrutura de rede, a infra-estrutura computacional, o sistema computacional, o domínio da aplicação e o ambiente físico.

Dey e Abowd (1999) propuseram então uma definição de contexto como sendo qualquer informação utilizada para caracterizar a situação de uma pessoa, lugar ou objeto relevante para a interação entre um usuário e uma aplicação,

incluindo até mesmo estes dois últimos. Neste caso, uma aplicação é dita sensível ao contexto quando utiliza informações de contexto a fim de fornecer serviço ou informação relevante de acordo com a tarefa do usuário.

Finalmente, Cho (2002) categorizou as ASCs quanto aos objetivos gerais da utilização de informações de contexto: (1) apresentação de informação contextual ou serviços ao usuário, como, por exemplo, uma aplicação que mostra as coordenadas GPS para seu usuário; (2) execução ou adaptação automática de um serviço de acordo com a informação contextual; e (3) identificação explícita da informação contextual para posterior recuperação e processamento.

As definições de Dey e Abowd (1999) e a categorização de ASCs proposta por Cho (2002) são utilizadas neste trabalho.

2.1.2. Serviços e APIs de MoCA

MoCA (Rubinsztejn et al, 2004) é um *middleware publish-subscribe* desenvolvido especialmente para atendimento de aplicações intradomínio, como, por exemplo, um campus universitário (Sacramento et al, 2004). Cada aplicação possui três tipos de elementos: servidores, *proxies* e clientes. Servidores e *proxies* são executados em nós de uma rede fixa; clientes são executados em dispositivos móveis. Em cada dispositivo móvel, é executado um programa monitor responsável por coletar dados referentes ao dispositivo e ao estado da rede.

Os serviços disponibilizados por MoCA são: *Context Information Service* (CIS), *Discovery Service* (DS), *Configuration Service* (CS) e *Location Inference Service* (LIS). O CIS é distribuído e cada um de seus servidores recebe e processa as informações de contexto do dispositivo móvel enviadas pelo correspondente monitor. O CIS também recebe as requisições para notificações ou subscrições e é responsável por gerar e enviar os eventos quando uma mudança em um estado do dispositivo for interessante para um subscritor.

Os outros serviços de MoCA complementam o CIS da seguinte maneira: (1) o DS armazena informações como nome, propriedades e endereço de aplicações ou serviços registrados no MoCA; (2) o CS armazena e gerencia as informações de configuração para todos os dispositivos móveis que podem usar os serviços de MoCA; e (3) o LIS é o serviço responsável por inferir a localização aproximada

de um dispositivo através da comparação do padrão de sinal de rádio-freqüência corrente do dispositivo com o padrão de sinal previamente medido em pontos de referência pré-definidos em uma área.

Além destes e outros serviços, MoCA também disponibiliza APIs para o desenvolvimento de ASCs. *Client* API e *Server* API permitem a implementação do cliente e do servidor de uma aplicação. *Communication Protocol API* permite implementar a comunicação síncrona entre os dispositivos. *Event-Based Communication Interface* (ECI) permite a implementação da comunicação assíncrona, isto é, através do registro de interesses.

O funcionamento geral dos serviços de MoCA é descrito através da seqüência de passos ilustrada na Figura 4.

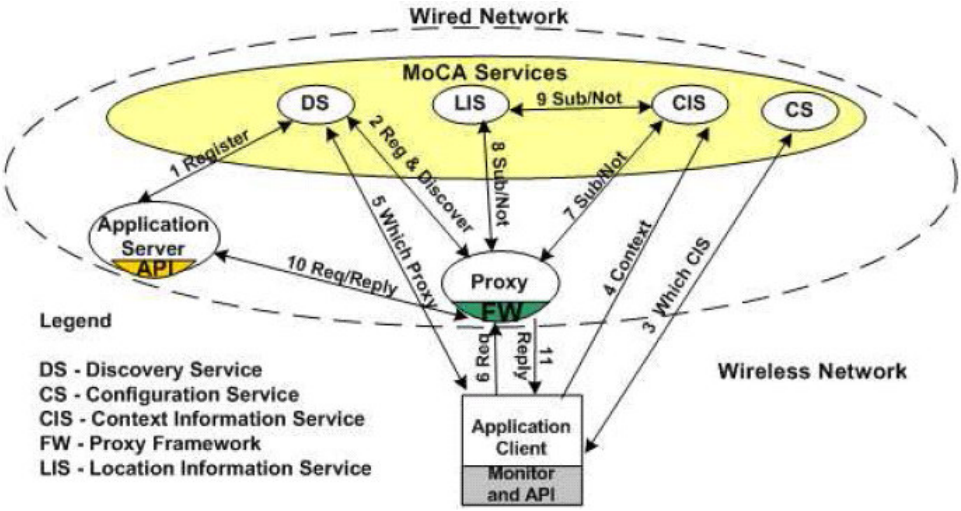

Figura 4. Arquitetura MoCA (Sacramento et al, 2004)

Nos passos 1 e 2 da Figura 4, o servidor e os *proxies* devem se registrar no DS, informando o nome e as propriedades do serviço que implementam. No passo 3, o monitor obtém do CS o endereço do CIS alvo e a periodicidade em que deve enviar as informações de contexto. No passo 4, o monitor periodicamente envia os dados do contexto do dispositivo móvel para o CIS. No passo 5, o cliente descobre um *proxy* que implementa o serviço através do DS. No passo 6, o cliente envia requisições para o *proxy* da aplicação, que processa a requisição do cliente, realiza adaptações necessárias e encaminha para o servidor. No passo 7, o *proxy* pode subscrever para o CIS com uma "expressão de interesse" o interesse em notificações sobre alterações de contexto de um cliente em particular.

Sempre que receber uma informação de contexto do dispositivo, o CIS verifica se este novo contexto é avaliado como verdadeiro em qualquer expressão de interesse. Caso afirmativo, o CIS gera uma notificação e a envia para todos os *proxies* que possuem interesse registrado na alteração de estado do dispositivo. Aplicações que pretendem receber informações sobre localização registram seus interesses no serviço LIS, como ilustrado no passo 8. No passo 9, o LIS é responsável por subscrever para o CIS o interesse em receber atualizações periódicas dos sinais de rádio-freqüência do dispositivo. Finalmente, no passo 10, quando o servidor recebe uma requisição de um cliente, a requisição é processada e uma resposta é enviada para alguns ou todos os *proxies*, que podem então processar a resposta de acordo com o novo estado do dispositivo móvel.

2.2. ASCs Baseadas em MoCA

A localização de um dispositivo, bem como dos serviços e dos outros dispositivos acessíveis a ele, mudam constantemente nos ambientes onde estão inseridas as ASCs. Tais aplicações precisam tratar de variações em seus contextos, tais como mudanças de temperatura e pressão, redução da bateria ou memória no dispositivo, mudanças na localização, orientação, velocidade ou capacidades do dispositivo, grau de proximidade a outros usuários e assim por diante.

Nesta seção, apresentamos dois exemplos de ASCs baseadas em MoCA: *Virtual Lines* e *Wireless Marketing Service*. Ambas foram desenvolvidas no LAC/PUC-Rio (LAC) e suas informações de contexto se referem basicamente à localização de dispositivos móveis. Tais aplicações foram selecionadas, porque suas funcionalidades nos permitiram obter uma primeira visão sobre pontos importantes a serem tratados em nosso trabalho.

2.2.1. Virtual Lines

Virtual Lines (VL) é uma aplicação sensível ao contexto desenvolvida para exemplificar o uso da arquitetura MoCA (LAC). Ela realiza o controle de filas virtuais em parques de diversão e tem como principal objetivo impedir que as pessoas esperem muito tempo nas filas ao mesmo tempo em que aproveitam da

melhor forma as atrações existentes em um parque. Ao passar próximo a uma atração, um dispositivo móvel pode coletar um *ticket* virtual correspondente a um lugar na fila. O sistema avisa ao usuário sobre a proximidade de sua vez a fim de que ele possa participar da atração. Quando o usuário não retorna a tempo, o sistema emite um alerta avisando que ele perdeu sua vez.

A aplicação VL está dividida em três módulos, tais como disponibilizados no sítio do LAC (LAC) e obedecendo à arquitetura geral de um ASC. Um primeiro módulo contém as classes de domínio da aplicação VL. Um segundo módulo representa a aplicação cliente, implementada para: (1) manter atualizada a informação sobre a situação dos usuários nas filas, (2) indicar ao usuário a atração mais próxima e (3) permitir a entrada de um usuário em uma fila virtual. Um terceiro módulo implementa a aplicação servidora, responsável por manter as reservas nas filas virtuais no parque.

A Figura 5 apresenta as classes usadas para representar o "negócio" do parque: atrações, filas, usuários, reservas, etc. Note que as classes de negócio são independentes das APIs de MoCA.

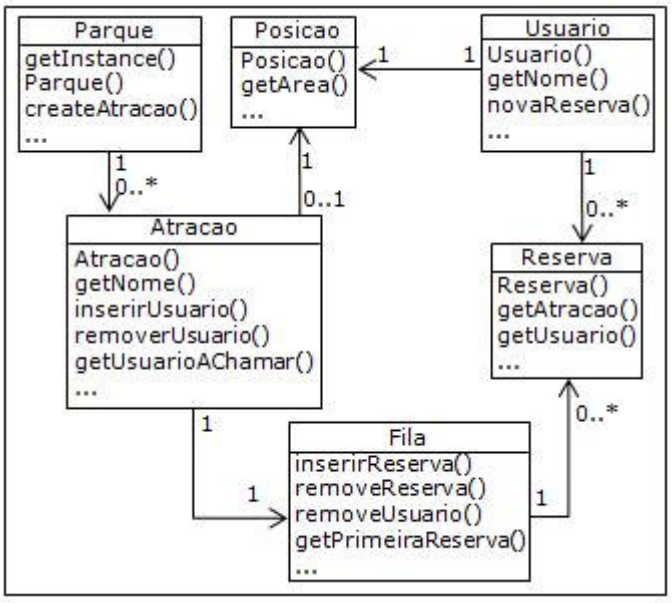

Figura 5. Classes de Negócio da Aplicação *Virtual Lines*

A Figura 6 apresenta as classes referentes à aplicação cliente de VL. A classe `FramePrincipal` implementa a interface com o usuário, permitindo visibilidade para a fila de atrações no parque e a solicitação pelo usuário de uma nova reserva na fila de uma atração. A classe `ListenerFilas` é responsável por

permanecer alerta quanto às filas de atrações no parque e é usada por `MainCliente` para a atualização das filas na interface da aplicação cliente. A classe `MainCliente` é também responsável por manter atualizada a posição de um usuário no parque através do serviço `LocationInferenceService` de MoCA. Note que, para implementar suas funcionalidades, as classes `ListenerFilas` e `MainCliente` utilizam serviços e APIs de MoCA.

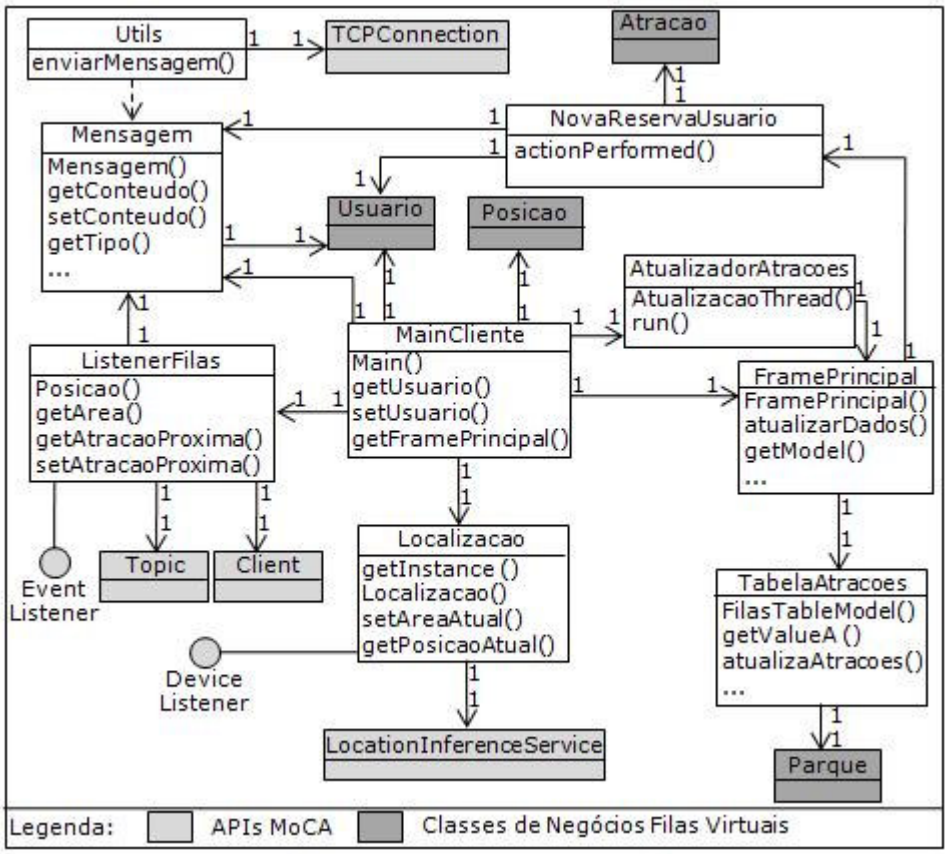

Figura 6. Módulo Cliente da Aplicação *Virtual Lines*

A Figura 7 apresenta o módulo de VL referente à aplicação servidora. Três classes principais são utilizadas. A classe `GerenciadorReservas` é responsável por: (1) inicializar uma *thread* de simulação das atrações; (2) manter a lista de usuários presentes no parque; e (3) efetuar as reservas de atrações para os usuários quando estes solicitarem a partir da aplicação cliente (Figura 6). A classe `PublicadorReservas` avisa os usuários sobre o *status* de suas reservas nas filas do parque. A classe `MainServidor` mantém o publicador de filas de atrações e uma conexão TCP para troca de mensagens. Note que, para implementar suas funcionalidades, `PublicadorReservas` e `MainServidor` utilizam *diretamente* as APIs de MoCA.

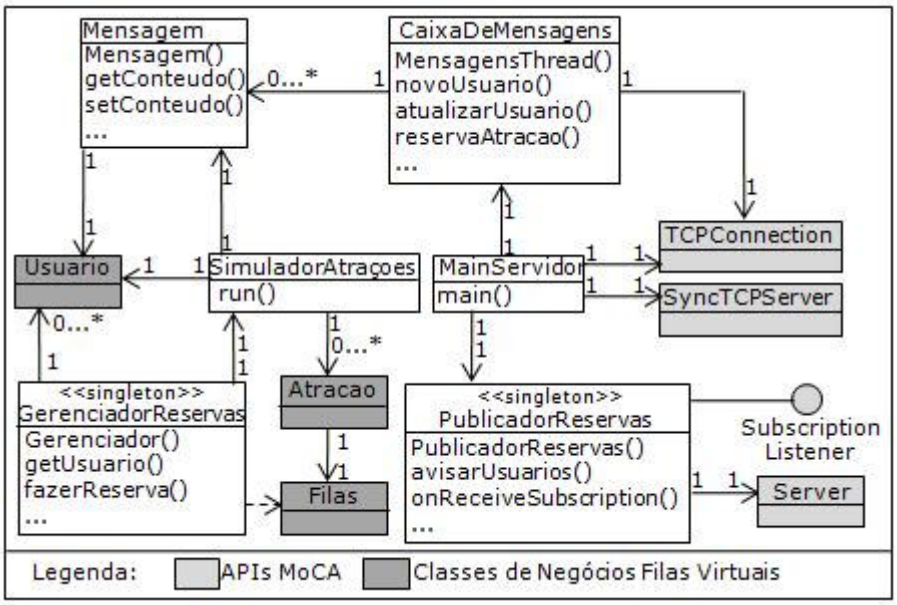

Figura 7. Módulo Servidor da Aplicação *Virtual Lines*

Na aplicação VL, a comunicação síncrona é utilizada para a comunicação do cliente (Figura 6) com o servidor (Figura 7) quando, por exemplo, um usuário solicita entrada em uma fila. Por outro lado, a comunicação assíncrona é usada quando o servidor precisa enviar alguma notificação ao cliente, por exemplo, quando o gerenciador de filas precisa avisar a um usuário que este deve comparecer a uma atração.

A Figura 8 ilustra um cenário que envolve comunicação síncrona para a realização da seguinte seqüência de passos: (1) a criação de um usuário e a configuração inicial de sua posição através do *Location Inference Service* de MoCA; (2) configuração da atração mais próxima do usuário para posterior uso em cenários referentes à reserva de atrações nas filas do parque.

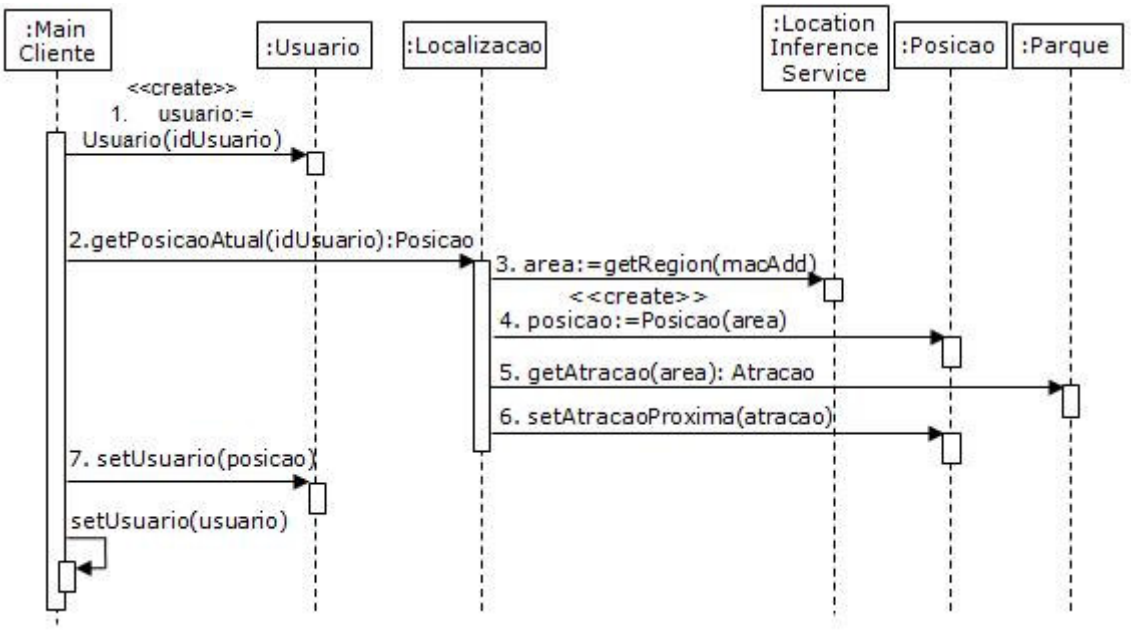

Figura 8. Cenário de Criação de Usuário em VL

A Figura 9 ilustra um cenário que envolve a solicitação de uma reserva por um usuário ao gerenciador de reservas. Note que as classes `FramePrincipal`, `NovaReservaUsuario`, `MainCliente` e `Utils` pertencem à aplicação cliente de VL (Figura 6); as classes `CaixaDeMensanges` e `GerenciadorReservas` pertencem à aplicação servidora (Figura 7); a classe `Usuario` pertence ao módulo de negócio (Figura 5) e é utilizada tanto pelo cliente quanto pelo servidor.

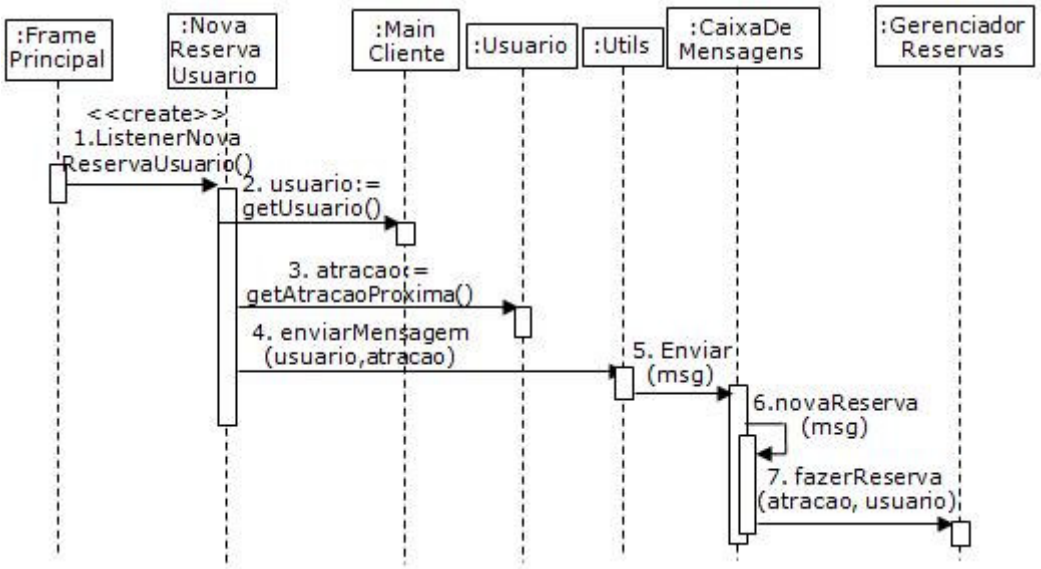

Figura 9. Cenário de Solicitação de uma Reserva em VL

A Figura 10 apresenta um trecho do código de *Virtual Lines* para exemplificar a manipulação de uma informação de contexto. Na linha 6, obtém-se o contexto de localização de um dispositivo através da chamada ao método `getArea()`. O contexto obtido serve de base para a definição das atrações e filas de um parque, as quais são tornadas disponíveis aos usuários daquele contexto (linhas 11 e 12). A partir da definição de atrações e filas, torna-se também possível disponibilizar serviços, como obter *ticket* para uma atração específica.

```
1: public Atracao getAtracaoMaisProximaDaPosicaoAtual (String
2: endMAC) //endMac:endereço MAC do dispositivo a ser localizado
3: {  String areaStr = "";
4:  try
5:  {//areaStr recebe a localização do dispositivo informado
6:    areaStr = lis.getArea(endMAC);
7:  }
8:  catch (LocationInferenceException e) {e.printStackTrace();}
9:  System.out.println("Cliente localizado na area: "+areaStr);
10: // Obtem a atração mais próxima da localização do cliente
11: Atração atracaoMaisProxima =
12: ParqueDeDiversoes.getInstance().getAtracao(areaStr);
13: // Retorna a posicao criada
14: return atracaoMaisProxima;
15: }
```

Figura 10. Manipulação de Contexto de Localização em *Virtual Lines*

Vale frisar que a Figura 10 apresenta a manipulação de apenas um exemplo de informação de contexto, isto é, a localização de um dispositivo em um parque de diversões. Entretanto, em ASCs geralmente são encontrados vários *outros* casos de manipulação de contextos.

2.2.2. Wireless Marketing Service

Wireless Marketing Service (WMS) é uma aplicação sensível ao contexto que possibilita a realização de campanhas de *marketing* (propaganda e promoções) de empresas através do envio de cupons para usuários de dispositivos móveis (LAC). WMS inclui componentes responsáveis pela gerência, resposta e

resgate dos cupons, além do cadastro e administração dos usuários, empresas e de suas campanhas de *marketing*. A Figura 11 ilustra a arquitetura de WMS.

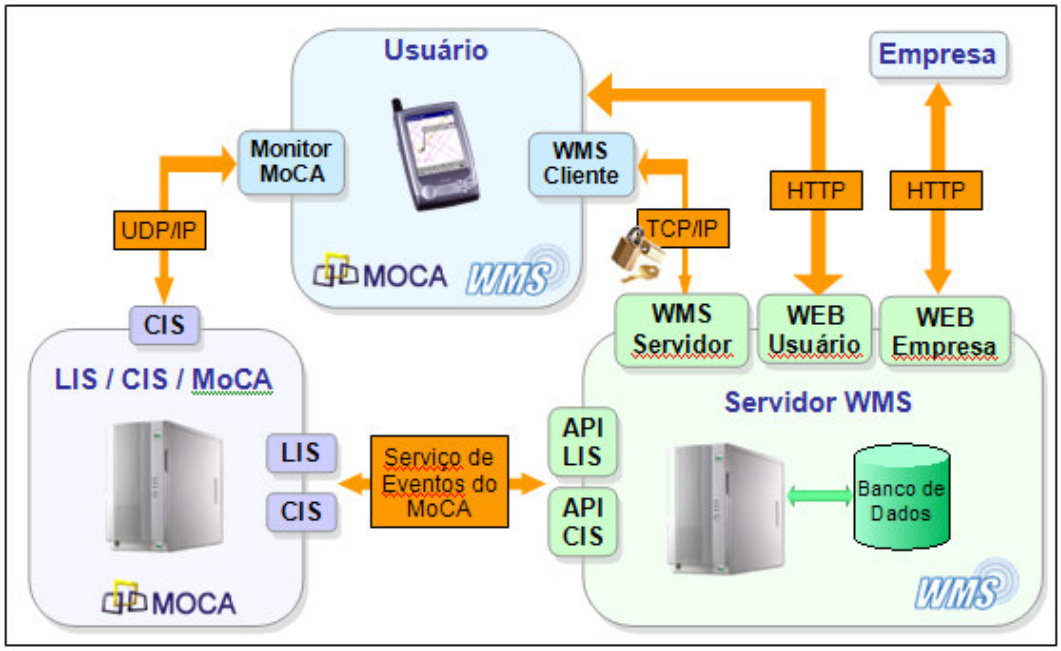

Figura 11. Arquitetura da Aplicação *Wireless Marketing Service* (LAC)

A aplicação WMS é composta por dois módulos principais (Figura 11). O módulo "Usuário" representa o usuário e seu dispositivo móvel; neste dispositivo, é necessário que estejam em execução dois aplicativos: o Monitor de MoCA (Seção 2.1.2) e o "WMS Cliente", responsável pela recepção e resgate dos cupons. O módulo "Servidor WMS" é responsável pelos serviços de envio e gerência dos cupons, cadastro e administração dos usuários, empresas e campanhas de *marketing*. Além dos módulos principais, note a representação dos serviços LIS e CIS de MoCA (Seção 2.1.2), utilizados para os serviços de sensibilidade ao contexto, como, por exemplo, a tarefa de localizar um usuário.

A Figura 12 apresenta o detalhamento do módulo "Servidor WMS". A classe `WMSRegister` é responsável pelo registro dos dispositivos de usuários. A classe `WMSSender` implementa a interface `DeviceListener` de MoCA a fim de receber a informação de localização dos usuários e assim possa lhes enviar os cupons de *marketing*. A classe `WMSResponse` implementa a interface `ConnectionListener` de MoCA a fim de receber a resposta dos usuários quando do envio de cupons. As classes `SyncTCPServer` e `ConnectionWaiter` de MoCA são utilizadas para o monitoramento das portas de comunicação com usuários.

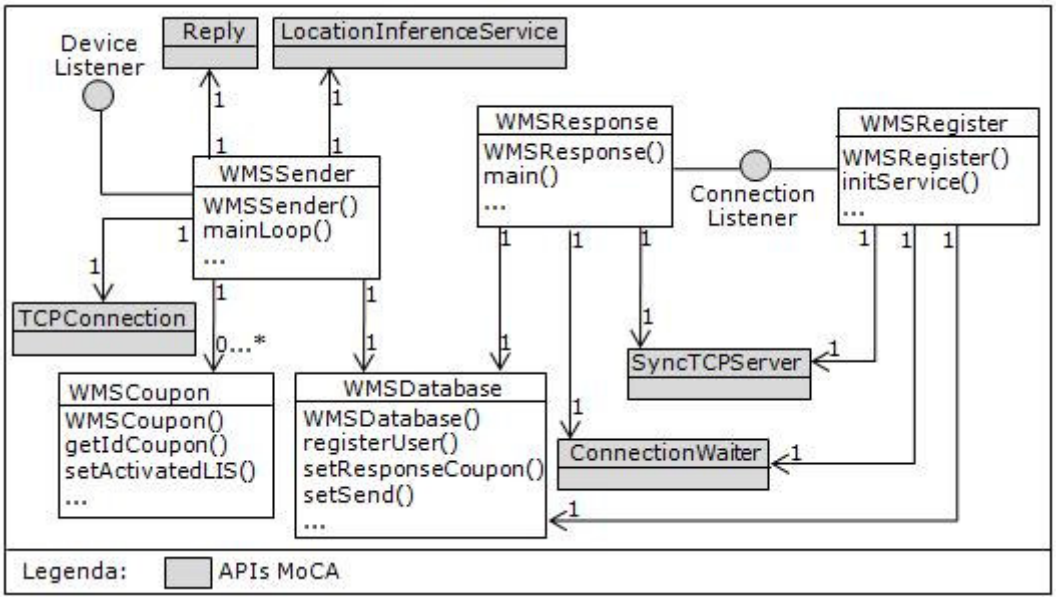

Figura 12. Módulo Servidor da Aplicação *Wireless Marketing Service*

A Figura 13 apresenta o diagrama de classes referente ao "WMS Cliente". A classe `WMSClient` é o elemento central, implementando a interface com o cliente e agregando: (1) o registro do dispositivo do usuário na aplicação "Servidor WMS" através de `WMSClientRegister`; (2) o recebimento de novos cupons de *marketing* através de `WMSClientNewCoupon`; e (3) a resposta dos usuários aos cupons recebidos através de `WMSClientCoupon`. A classe `WMSCouponsModel` corresponde ao modelo de cupom usado pela interface `WMSClient`.

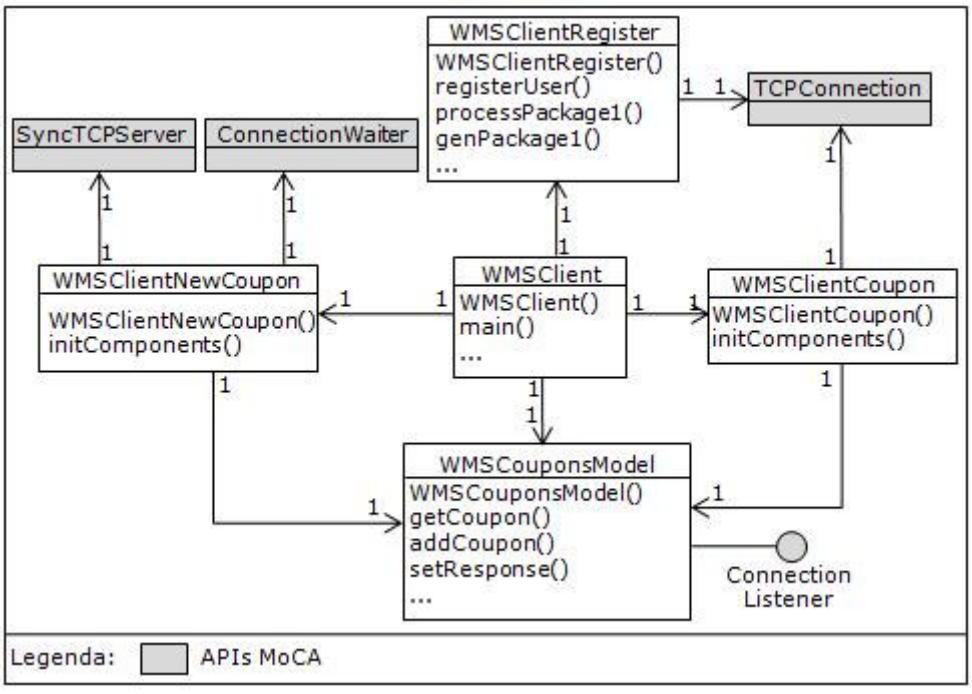

Figura 13. Módulo Cliente da Aplicação *Wireless Marketing Service*

A Figura 14 ilustra a seqüência de interações correspondentes ao cenário de notificação de mudança de localização de um usuário. Primeiramente, o *Location Inference Service* informa ao `WMSSender` a nova região onde está localizado o usuário. O `WMSSender`, por sua vez, recupera todos os cupons de *marketing* da nova área, para enviar ao usuário através do método `sendCoupon()`. A seguir, `WMSSender` atualiza o banco para cumprimento do gerencimento de cupons.

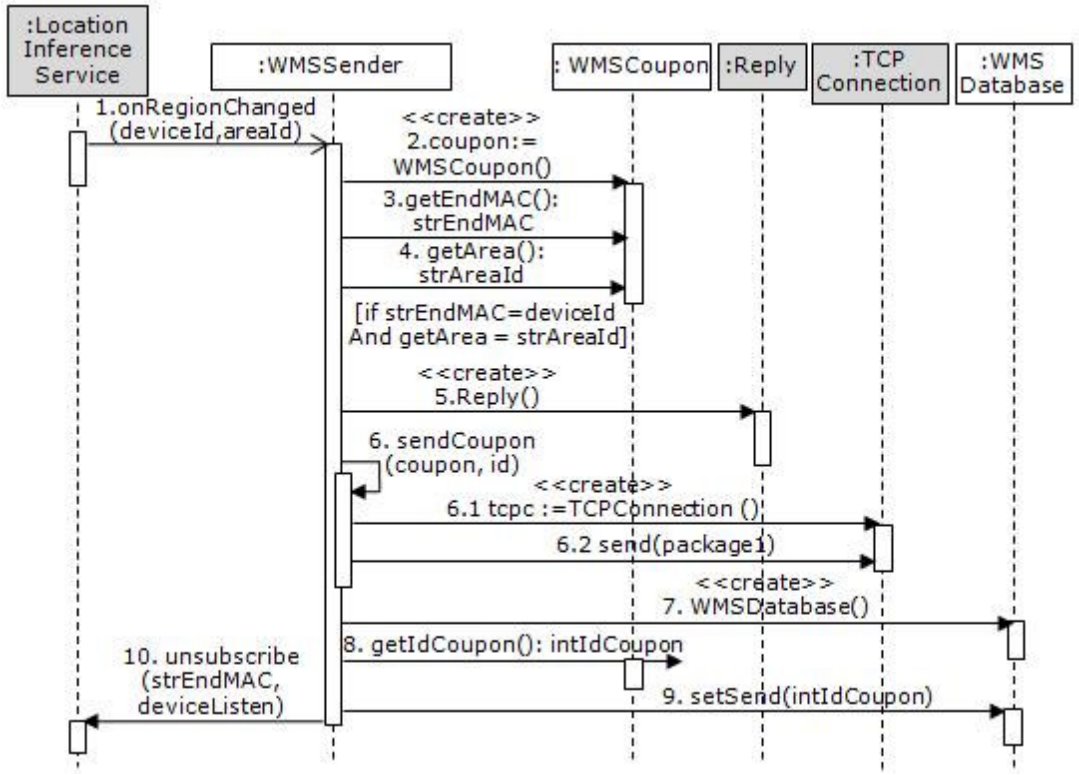

Figura 14. Cenário de Notificação de Localização de Usuário em WMS

2.3. Análise do Uso Direto das APIs de Middlewares em ASCs

Do ponto de vista da Engenharia de Software e, particularmente, das questões relacionadas à modularidade e reuso, várias limitações emergem do uso direto de APIs de *middlewares* no desenvolvimento de ASCs. Apresentamos uma breve análise de tais limitações a seguir.

Em primeiro lugar, quando ASCs como *Virtual Lines* e *Wireless Marketing Service* (Seção 2.2) utilizam diretamente as APIs disponibilizadas por *middlewares*, tais aplicações perdem em "variabilidade". Mais especificamente: as ASCs que fazem uso *direto* destas APIs são fortemente acopladas ao paradigma de coordenação adotado pelo *middleware* sendo utilizado. No caso do uso das

APIs de MoCA (Seção 2.1.2), as ASCs *dependem* do paradigma *publish-subscribe* ou baseado em eventos.

De fato, no caso da necessidade de evolução de VL e WMS (Seção 2.2) para o uso de outros *middlewares* que, além de oferecerem suporte ao desenvolvimento de ASCs, tratam também de outras requisitos importantes como, por exemplo, o tratamento de exceções, as ASCs necessariamente deverão ser *reimplementadas* para uso de outros paradigmas, como aquele baseado em espaços de tuplas (Iliasov & Romanovsky, 2005).

Além disso, a mudança de *middleware* poderia ser motivada por limitações mais granulares relacionadas, por exemplo, à técnica adotada na modelagem de contexto. Por exemplo, MoCA utiliza o modelo de pares atributo-valor, que utiliza um atributo para descrever uma propriedade específica do contexto. Este modelo é o mais elementar e de mais simples implementação.

Entretanto, se outros modelos devessem ser utilizados por conta da ausência de tipagem e inter-relacionamento entre contextos, as ASCs mais uma vez teriam que ser *reimplementadas*. Isto aconteceria, por exemplo, se precisássemos usar modelos baseados em esquemas, lógica ou ontologias, como fazem outros *middlewares* (Strang & Linnhoff-Popien, 2004).

Além do mais, outras limitações são detectadas quando comparamos e analisamos mais de perto a estrutura e a dinâmica de ASCs (Seção 2.2). Verificamos que o nível de reuso poderia ser facilmente aumentado se as funcionalidades recorrentes fossem abstraídas, mesmo para o uso de um *middleware* específico, como é o caso de MoCA em VL e WMS.

Por exemplo, as ASCs que utilizam MoCA seguem um mesmo padrão arquitetural, incluindo *sempre* uma aplicação cliente e outra servidora. Nos dois tipos de aplicação, podemos abstrair muitas funcionalidades recorrentes e abrangentes, relacionadas à manipulação de contextos, adaptação diante da ocorrência de eventos, troca de mensagens entre aplicações e interface gráfica.

Por fim, verificamos também que o uso de alguns serviços de MoCA poderiam ser concentrados em apenas uma das aplicações. Por exemplo, em VL, a aplicação cliente original utiliza o LIS; porém, adotando o uso de novas abstrações e mecanismos de decomposição nestas aplicações (como os de ESSMA), facilmente podemos chegar à "transferência" total do uso do LIS para a aplicação servidora, como já acontece originalmente em WMS.