2
Sensibilidade ao Contexto

Sensibilidade ao contexto é a habilidade dos dispositivos móveis terem conhecimento do estado e do ambiente físico ao redor do usuário. Aplicações ou serviços sensíveis ao contexto são desenvolvidos, por exemplo, quando telefones móveis podem inspecionar o ambiente, ao invés de apenas a rede possuir informações sobre os dispositivos. Outros exemplos de aplicações móveis sensíveis ao contexto podem utilizar informações como perfil de usuários, localização, altitude, orientação, temperatura, pressão sanguínea, velocidade, memória disponível, bateria do dispositivo, e assim por diante. A utilização destas informações permite que uma aplicação tenha conhecimento de sua "situação" em um dado momento. Em outras palavras, aplicações sensíveis ao contexto possuem a habilidade de interpretar e utilizar o contexto de uma situação, o que pode servir de base para um comportamento adaptativo.

As aplicações são sensíveis ao contexto a fim de fornecer uma qualidade razoável de serviços a seus usuários (Capra et al. 2001). Entretanto, de modo geral, estas aplicações ainda são inadequadas aos ambientes distribuídos móveis e, freqüentemente, não são capazes de tolerar um ambiente de execução que muda rapidamente, ou mesmo de tirar vantagem da disponibilidade de novos serviços. Isto ocorre porque a sensibilidade ao contexto exige novos métodos de exploração das capacidades dos dispositivos móveis. Além disso, há uma grande potencialidade destes dispositivos em trazer para o mercado um novo conjunto de aplicações que ainda não foram experimentados fora do âmbito de pesquisa.

Neste capítulo, são apresentadas inicialmente algumas definições gerais sobre sensibilidade ao contexto, tais como contexto, tipos de contexto e aplicações sensíveis ao contexto. Em seguida, são apresentados os *middlewares* para desenvolvimento de aplicações sensíveis ao contexto, sua estrutura geral, alguns modelos de comunicação e exemplos.

2.1.
Definições Básicas

Várias tentativas foram realizadas no esforço de definir contexto e aplicações sensíveis ao contexto e tais tentativas enfatizaram quase que exclusivamente a localização de dispositivos móveis. A seguir, apresentamos um breve histórico da evolução de definições sobre sensibilidade ao contexto, como descrito em Mitchell (2002a).

2.1.1.
Contexto e Tipos de Contexto

Active Maps (Schilit & Theimer, 1994a) foi o primeiro trabalho a utilizar explicitamente o conceito de sensibilidade ao contexto. Como contexto foram utilizadas informações como localização, identificação de pessoas e objetos próximos, bem como mudanças que ocorrem sobre estes objetos ao longo do tempo. 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.

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: (i) usuário – hábitos, estado mental ou características fisiológicas; (ii) ambiente social – proximidade de outros, relacionamento social e tarefas colaborativas; e (iii) tarefas – objetivos de atividades ou objetivos gerais do usuário. A categoria ambiente físico possui as subcategorias: (i) localização – absoluta (coordenada GPS) ou relativa (dentro de uma dada sala); (ii) infra-estrutura – a computação ao redor e interações no ambiente; e (iii) condições – condições físicas típicas do ambiente, tais 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 argumentaram que um dispositivo móvel opera em um contexto bastante amplo, que inclui a infra-estrutura de rede e a infra-estrutura computacional, o sistema computacional, o

domínio da aplicação e o ambiente físico. Finalmente, Dey e Abowd (1999) propuseram 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. Esta é a definição de contexto utilizada neste trabalho.

2.1.2. Aplicações Sensíveis ao Contexto

Inicialmente, as aplicações sensíveis ao contexto foram definidas como sendo aplicações que são informadas sobre o contexto ou que se adaptam ao contexto (Schilit & Theimer, 1994a). Posteriormente, Hull et al. (1997) e Pascoe (1998) definiram mais explicitamente uma aplicação sensível ao contexto. 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. A seguir, Dey e Abowd (1999) apresentaram outra definição, segundo a qual 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.

Segundo Cho (2002), por suas características quanto à utilização do contexto, as aplicações sensíveis ao contexto podem ser categorizadas em: (i) apresentação de informação contextual ou serviços ao usuário, como por exemplo, uma aplicação que mostra as coordenadas GPS para seu usuário; (ii) execução automática de um serviço, ou ações que são engatilhadas de acordo com a informação contextual (Schilit et al., 1994), ou ainda, adaptação contextual (Pascoe, 1998), isto é, trata-se da habilidade de determinadas aplicações em executar ou modificar um serviço automaticamente baseado no contexto corrente; e (iii) identificação explícita da informação contextual para posterior recuperação e processamento (Pascoe, 1998) ou a habilidade de certas aplicações em associar dados digitais com o contexto de um usuário.

2.2.
Computação Distribuída Tradicional e Móvel

Assim como em sistemas distribuídos tradicionais, uma arquitetura para aplicações sensíveis ao contexto também pode ser definida a partir de camadas similares às camadas do modelo OSI (Capra et al., 2001). A divisão em camadas possibilita uma abstração horizontal em que a tecnologia em uma determinada camada é independente daquelas utilizadas nas camadas inferiores. De modo geral, as camadas existentes são:

(i) aplicação: camada clássica na definição do modelo OSI;

(ii) *middleware*: interface entre a aplicação e o sistema operacional. Possibilita um alto nível de abstração, escondendo grande parte da complexidade associada à distribuição, como heterogeneidade, escalabilidade, compartilhamento de recursos e tolerância a falhas;

(iii) inter-rede: inclui todas as funções de baixo-nível, as quais correspondem às camadas inferiores como transporte, rede, etc.

Na computação distribuída tradicional, ocorreu um grande crescimento no desenvolvimento de aplicações em função do sucesso de alguns *middlewares* em facilitar a comunicação entre os componentes distribuídos (ex. CORBA, Java/RMI e MQSeries). Foram oferecidas aos engenheiros de aplicação abstrações de comunicação que reduziram a complexidade no tratamento de questões, como a localização de componentes distribuídos, falhas de rede e heterogeneidade do hardware. Os *middlewares* da computação distribuída tradicional são baseados no "princípio da transparência" (Emmerich apud Capra et al., 2003), isto é, detalhes de implementação são escondidos do usuário bem como de projetistas de aplicações e permanecem encapsulados em uma camada intermediária.

Embora projetados e utilizados com grande sucesso em sistemas distribuídos tradicionais, os *middlewares* citados acima ainda não oferecem características apropriadas quando consideramos os cenários da computação móvel. Isto ocorre em função de três razões principais (Capra, et al. 2001). Primeiro, porque as primitivas de interação, como transações distribuídas, requisições de objetos ou chamadas a procedimentos remotos, assumem a existência de conexão constante e grande largura de banda, quando na verdade os

sistemas móveis possuem conexão instável e largura de banda restrita. Segundo, os *middlewares* orientados a objetos suportam principalmente a comunicação síncrona ponto-a-ponto, enquanto nos ambientes móveis é freqüente a situação em que os dispositivos cliente e servidor não estão conectados ao mesmo tempo. Terceiro, os sistemas distribuídos tradicionais assumem a existência de um ambiente de execução estacionário, o que contrasta com os novos cenários extremamente dinâmicos de aplicações móveis. Anteriormente, era razoável esconder completamente de uma aplicação as informações de contexto, como localização e detalhes de implementação. Agora isso se torna mais difícil e faz pouco sentido preservar tal decisão, considerando os altos níveis de dinamicidade e heterogeneidade intrínsecos de ambientes móveis.

De fato, para Capra et al. (2003), a transparência não pode ser considerada como o princípio fundamental para o desenvolvimento de abstrações e mecanismos de *middlewares* da computação móvel. Pela característica de transparência, os *middlewares* precisam tomar decisões em nome das aplicações. Uma aplicação, entretanto, pode tomar decisões muito mais eficientes e com muito mais qualidade se for baseada em informações específicas de seu contexto. Pelas mesmas razões, a sensibilidade ao contexto também possibilitaria a um *middleware* executar de forma mais eficiente. Em resumo, uma vez que os *middlewares* tradicionais não consideram as necessidades específicas de aplicações móveis sensíveis ao contexto, torna-se necessário o desenvolvimento de *middlewares* específicos para o desenvolvimento de aplicações móveis. Tais *middlewares* são descritos em detalhes na seção a seguir.

2.3. *Middlewares* para Aplicações Sensíveis ao Contexto

Middlewares específicos para o desenvolvimento de aplicações sensíveis ao contexto devem ser capazes de realizar três tarefas principais a fim de permitir a adaptação baseada no contexto:

- coleta das informações de contexto;
- disseminação das informações de contexto, e
- notificação de alterações de contexto de acordo com o interesse dos clientes, quando é disparada a adaptação desejada.

Para realizar as tarefas acima com eficiência e escalabilidade, as arquiteturas de *middlewares* para aplicações sensíveis ao contexto são, em geral, baseadas em alguns dos seguintes elementos: sensores, provedores de informação, *brokers* e consumidores de informação. *Sensores* são os elementos responsáveis por capturar as informações do ambiente e transformá-las em um formato digital que possa ser processado pelo ambiente de computação. *Provedores* ou *produtores de informação de contexto* são responsáveis por popular o sistema com as informações de contexto, geradas a partir de outras informações de contexto, ou a partir de dados recebidos por sensores. *Brokers* de contexto, ou serviço de contexto, são responsáveis pelo registro de interesses e a disseminação das informações de contexto. *Consumidores* são os elementos que possuem interesse em informações de contexto. Consumidores registram interesses por informações de contexto ou por eventos baseados em contexto e são notificados pelo *broker* quando a condição de interesse ocorre. Consumidores de contexto podem ser aplicações, serviços ou outros componentes de um *middleware*.

Alguns *middlewares* facilitam o desenvolvimento e a integração dos sensores por meio de adaptadores, que implementam a interface entre o sensor e os *brokers* de contexto. Além disso, muitas vezes o papel do sensor se confunde com o do provedor de informação de contexto que, por sua vez, deve conhecer muito bem o modelo adotado para o contexto a ser gerado, produzir informações de contexto consistentes e alimentar o *broker* com o contexto produzido. Provedores de contexto também podem atuar como consumidores de contexto, sobretudo quando implementam a inferência de contexto. As subseções a seguir descrevem em detalhes diferentes aspectos dos *middlewares* que facilitam o desenvolvimento de aplicações sensíveis ao contexto.

2.3.1. Modelagem de Contexto

Os modelos de contexto adotados pelos diversos *middlewares* existentes são classificados por Strang e Linnhoff-Popien (2004) em: (i) modelos de pares atributo-valor, (ii) modelos baseados em esquemas, (iii) modelos gráficos, (iv) modelos orientados a objetos, (v) modelos baseados em lógica e (vi) modelos baseados em ontologias.

Um modelo de pares atributo-valor utiliza pares atributo-valor para representar os contextos, onde um atributo descreve uma propriedade específica do contexto. É o modelo mais elementar e de mais simples implementação, em função da ausência de tipagem e inter-relacionamento entre contextos; por este motivo, foi utilizado em vários projetos de computação sensível ao contexto.

Os modelos baseados em esquemas utilizam estruturas de dados hierárquicas, compostas de pares atributo-valor e descritas por linguagens de marcadores como dialetos de XML. A informação de contexto possui uma estrutura bem definida e permite a composição de informações.

Modelos gráficos utilizam linguagens gráficas para modelar o contexto. Embora capazes de modelar ricas composições e inter-relacionamentos entre informações de contexto, estes modelos são limitados quanto à modelagem do comportamento dinâmico do contexto.

Os modelos orientados a objetos fazem a representação de contextos por meio de modelos de objetos, usando encapsulamento, reutilização e abstração. Por sua vez, modelos baseados em lógica utilizam fatos, expressões e regras para especificar como uma informação de contexto pode ser inferida ou derivada a partir de outra.

Finalmente, modelos baseados em ontologias utilizam ontologias para descrever contextos e seus inter-relacionamentos. Tais modelos possuem a capacidade de descrever contextos complexos, sendo atualmente objeto de interesse crescente em vários projetos (Ranganathan & Campbell, 2003).

2.3.2. Paradigmas de Coordenação

Middlewares baseados em passagem de mensagens, como CORBA ou Java RMI, são úteis para o desenvolvimento de sistemas distribuídos tradicionais (Gaddah & Kunz, 2003). O modelo de coordenação adotado por estes *middlewares* é baseado no padrão *request/reply*, onde um objeto espera passivo até que executem uma operação sobre ele. Este tipo de modelo é adequado para uma rede local com pequeno número de clientes e servidores, mas não é escalável para grandes redes como a Internet. A principal razão para isto é a natureza síncrona do modelo *request/reply*, que suporta somente a comunicação um-para-

um e impõe um forte acoplamento entre os participantes, que precisam conhecer com quem irão se comunicar.

Portanto, um modelo baseado em mensagens é inapropriado para ambientes dinâmicos e com problemas de conectividade, como é o caso de aplicações móveis sensíveis ao contexto. Diversos paradigmas de coordenação têm sido adotados para a implementação de *middlewares* de aplicações da computação móvel. A seguir, são descritos os paradigmas de coordenação mais conhecidos.

Baseados em Eventos ou *Publish-Subscribe*

O paradigma baseado em eventos ou paradigma *publish-subscribe* é uma alternativa possível para o tratamento de aplicações móveis em larga escala (Bacon et al. *apud* Gaddah & Kunz, 2003). Eventos contêm dados que descrevem uma requisição ou mensagem e são propagados dos componentes emissores, denominadores *publicadores*, para os componentes receptores, denominados *subscritores*. Os publicadores são os agentes que enviam informação a um componente central, enquanto os subscritores expressam seu interesse no recebimento de eventos particulares. O *broker* é o componente central responsável por registrar todas as subscrições, comparar as publicações com todas as subscrições e notificar os subscritores interessados. Arquiteturas baseadas em eventos ou *publish-subscribe* oferecem um modelo de coordenação com fraco acoplamento entre os publicadores e subscritores e onde a notificação assíncrona de eventos é naturalmente suportada.

Baseados em Espaço de Tuplas

Como discutido anteriormente, modelos de coordenação baseados em passagem de mensagens, possuem a desvantagem de acoplamento forte entre os participantes. Nestes modelos, é necessário que o emissor tenha conhecimento da identidade exata e do endereço do receptor. Além disso, existe a necessidade de sincronização, isto é, o emissor precisa esperar até que o receptor esteja pronto para trocar informações. Em sistemas abertos, esta característica é bastante restritiva. É necessária a utilização de um estilo de computação mais desacoplado. A computação deve prosseguir mesmo na presença de desconexão e a conectividade deve ser explorada sempre que se tornar disponível. Uma

alternativa para evitar o problema de forte acoplamento é o conceito de espaço de tuplas (Gelernter, 1985).

Um espaço de tuplas é espaço de memória compartilhado globalmente e distribuído por todos os processos e *hosts* participantes. Os processos que usam este modelo se comunicam gerando tuplas e antituplas que são submetidas ao espaço de tuplas. Tuplas são dados estruturados tipados, como por exemplo, objetos em C++ e Java, onde cada tupla é formada por uma coleção de campos de dados tipados e representa uma parte coesa da informação. Em um sistema baseado em espaço de tuplas, todas as comunicações interprocesso são exclusivamente realizadas utilizando o espaço de tuplas e qualquer processo que usa um espaço de tuplas possui a habilidade para acessar todas as tuplas que o espaço contém, inserir dinamicamente novas tuplas, encontrar associações para antituplas não destrutivas e remover tuplas gerando antituplas destrutivas associadas (Gelernter, 1985). Sistemas baseados em espaços de tuplas provaram sua habilidade para facilitar a comunicação em configurações sem fio. Esta forma de comunicação é bastante adequada às configurações móveis onde estão envolvidas a mobilidade lógica e a física (Gaddah & Kunz, 2003).

Reflexão Computacional

A reflexão computacional (Smith, 1982) foi inicialmente utilizada na área de linguagens de programação a fim de prover suporte ao projeto de linguagens mais extensíveis e abertas (Maes, 1987). Em seguida, também foi aplicada em outras áreas, incluindo sistemas operacionais e sistemas distribuídos. O princípio da reflexão permite que um programa acesse e mude seu próprio comportamento (Maes, 1987). Portanto, um *middleware* reflexivo (Blair, 1998; Tripathi, 2002) explora os mecanismos da reflexão computacional para implementar mobilidade e serviços sensíveis ao contexto. A reflexão é usada para monitorar a reconfiguração interna do *middleware* (Roman et al., 2001). Um *middleware* reflexivo é dividido em dois níveis: nível-base e meta-nível. O nível-base representa o *middleware* e o núcleo da aplicação. O meta-nível contém os blocos responsáveis por oferecer suporte à reflexão. Estes dois níveis estão conectados através de um protocolo de meta-objetos a fim de garantir que as modificações no meta-nível sejam refletidas em correspondentes modificações no nível-base. Desta

forma, modificações no núcleo da aplicação devem ser refletidas no meta-nível. Os elementos do nível-base e do meta-nível são respectivamente representados por objetos do nível-base e objetos do meta-nível. A principal motivação da abordagem reflexiva é tornar o *middleware* mais adaptável ao seu ambiente, isto é, mais capaz de lidar com mudanças (Gaddah & Kunz, 2003). A reflexão é considerada uma técnica poderosa para construir *middlewares* que oferecem suporte ao desenvolvimento de aplicações móveis sensíveis ao contexto (Capra, et al. 2003).

2.4. Exemplos de *Middlewares*

Nesta seção, são apresentados exemplos de *middlewares* para desenvolvimento de aplicações sensíveis ao contexto. Para cada um dos paradigmas de coordenação descritos (Seção 2.3.2) é discutido um exemplo representativo. Desta forma, MoCA (Seção 2.4.1) ilustra o paradigma *publish-subscribe*, CAMA (2.4.2) o baseado em espaço de tuplas e CARISMA (2.4.3) a reflexão computacional.

2.4.1. MoCA

MoCA é um *middleware* que oferece suporte ao desenvolvimento e execução de aplicações colaborativas sensíveis ao contexto para usuários móveis (Sacramento et al., 2004). Sua infra-estrutura é baseada em uma rede sem fio (802.11) com usuários conectados através de dispositivos móveis, como *laptops* ou *palmtops*. Este *middleware* foi desenvolvido especialmente para atender aplicações intradomínio, como por exemplo, um campus universitário. MoCA possui uma Interface de Comunicação baseada em Eventos (ECI), que fornece facilidades para implementar a comunicação assíncrona usando a abordagem *publish-subscribe*.

MoCA é composto basicamente por APIs cliente e servidor, um conjunto de serviços centrais para realizar o monitoramento e a inferência de contextos dos dispositivos móveis e um *framework* orientado a objetos para instanciar os *proxies* para aplicação. Cada aplicação possui três tipos de elementos: servidores, *proxies*,

e clientes. Os dois primeiros são executados em nós de uma rede estática; os clientes são executados em dispositivos móveis. Em cada dispositivo móvel, é executado um programa monitor, o qual é responsável pela coleta de dados referentes ao dispositivo e ao estado da rede. Os principais 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 que foram enviadas pelo correspondente monitor. O CIS também recebe as requisições para notificações ou subscrições e é responsável por gerar e enviar os eventos sempre que uma mudança em um estado do dispositivo for interessante para um subscritor. CIS é um *broker* (Seção 2.4.1) que trata do contexto mais elementar de dispositivo. Na arquitetura MoCA, quando um novo contexto é necessário, deve ser implementado um novo serviço para atuar como provedor, *broker* e consumidor dos eventos que devem ser interpretados.

Os outros serviços de MoCA complementam o CIS da seguinte maneira: (i) o DS armazena informações como nome, propriedades e endereço de aplicações ou serviços registrados no MoCA; (ii) o CS armazena e gerencia as informações de configuração para todos os dispositivos móveis que podem usar os serviços centrais de MoCA; e (iii) 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.

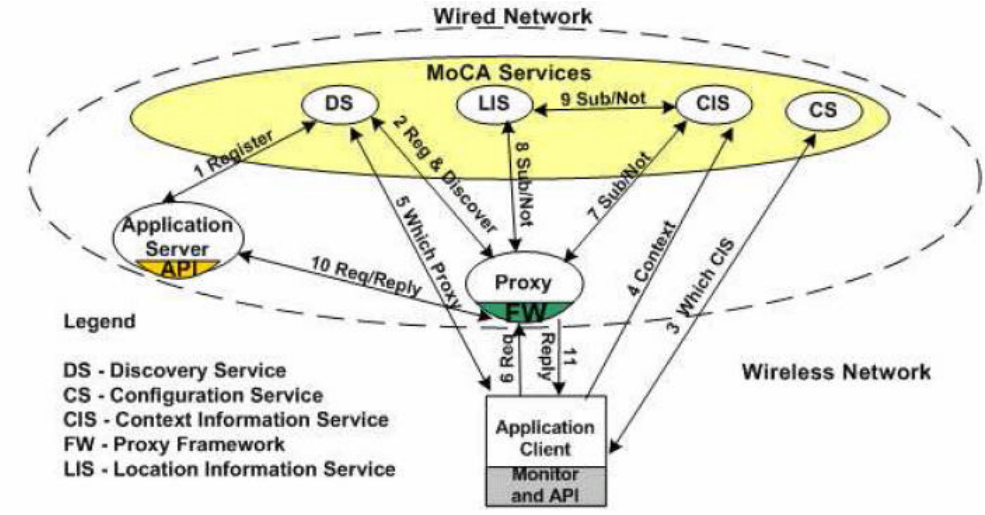

Figura 1. Arquitetura MoCA (Sacramento et al., 2004)

Um cenário que descreve o funcionamento geral de MoCA pode ser descrito através da seqüência de passos ilustrada na Figura 1. Nos passos 1 e 2, o servidor da aplicação e os *proxies* devem se registrar no DS, informando o nome e as propriedades do serviço colaborativo que implementam. No passo 3, o monitor, quando iniciado, 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 colaborativo desejado através do DS. No passo 6, o cliente envia requisições para o *proxy* da aplicação, que processa a requisição do cliente, realiza as adaptações específicas e encaminha para o servidor da aplicação. No passo 7, o *proxy* pode subscrever para o CIS com uma "expressão de interesse" o interesse em notificações sobre mudanças de contexto de um cliente em particular. Sempre que o CIS receber uma informação de contexto do dispositivo, ele 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 nesta mudança de estado do dispositivo. Aplicações que desejam 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 da aplicação recebe uma requisição de um cliente, a requisição é processada e uma resposta é enviada de volta para alguns ou todos os *proxies*, que podem então se adaptar ou processar a resposta de acordo com o novo estado do correspondente dispositivo móvel em sua conexão sem fio.

MoCA possui diversas APIs úteis para o desenvolvimento de aplicações sensíveis ao contexto. *Client* API e *Server* API permitem a implementação do cliente e do servidor de uma aplicação sensível ao contexto. *Communication Protocol API* permite implementar a comunicação síncrona. *Event-Based Communication Interface* (ECI) permite a implementação da comunicação assíncrona através da abordagem *publish-subscribe*. A Figura 2 ilustra as principais dependências entre as API da arquitetura MoCA. Note que os serviços CIS e LIS utilizam ambas as formas de comunicação, permitido que a consulta de informações de contexto seja realizada de forma síncrona ou através do registro de

interesses. Além disso, a Figura 2 permite observar que o serviço LIS depende dos serviços SRM e CIS para seu correto funcionamento.

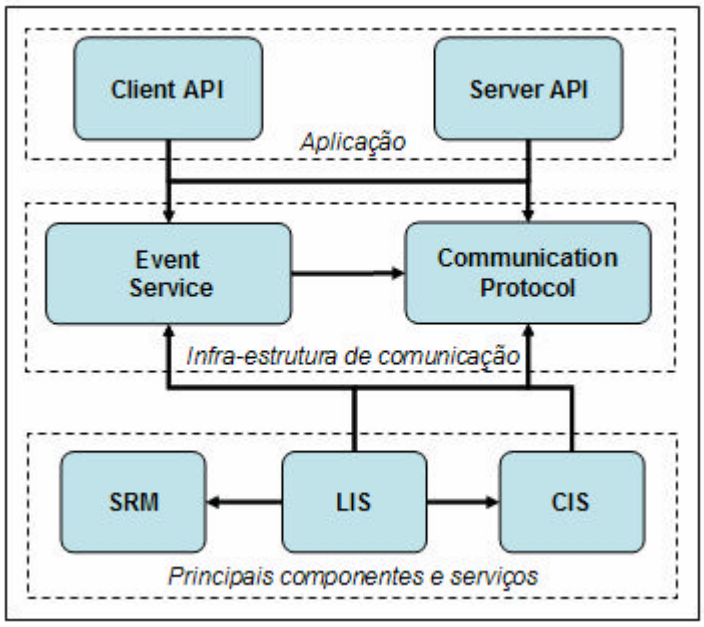

Figura 2. Dependências entre as APIs de MoCA

2.4.2. CAMA

CAMA é um *framework* para o desenvolvimento de aplicações móveis usando o paradigma de agentes (Iliasov & Romanovsky, 2005a). Este *framework* fornece um conjunto de abstrações, juntamente com um *middleware* e uma camada de adaptação que permitem aos desenvolvedores tratar características de aplicações móveis como abertura, tolerância a falhas, comunicação assíncrona e anônima, além de mobilidade de dispositivo e de código.

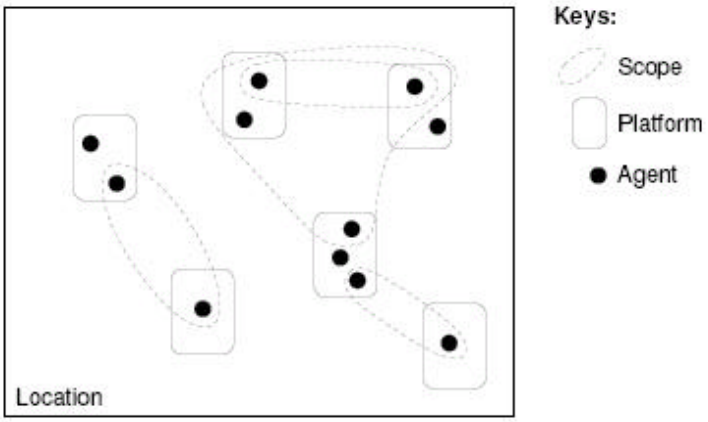

Figura 3. Principais Abstrações de CAMA (Iliasov & Romanovsky, 2005a)

CAMA suporta o desenvolvimento disciplinado de sistemas através de estruturações recursivas que utilizam as abstrações de localização, escopo, agente e papel. A Figura 3 ilustra a relação entre as abstrações utilizadas em CAMA.

CAMA especifica um conjunto de abstrações, como *localização*, *escopo*, *agente* e *plataforma*. Qualquer sistema CAMA consiste em um conjunto de *localizações*. Uma localização é um repositório para os *escopos*. Um escopo fornece um espaço de coordenação dentro do qual agentes compatíveis podem interagir. *Agentes* são as entidades ativas do sistema. Cada agente é executado em uma *plataforma* e vários agentes podem residir em uma única plataforma. Uma plataforma provê um ambiente de execução para os agentes e uma interface para o *middleware* de localização.

Em CAMA, uma localização pode ser associada a uma localização física particular, tal como um teatro, armazém ou sala de reunião, e pode ter certas restrições quanto aos tipos de escopos suportados. Localização é o elemento central do sistema, uma vez que fornece um meio de comunicação e coordenação entre os agentes. Assume-se que cada localização tem um nome único que em geral corresponde ao endereço IP do *host* na rede onde a localização reside. Uma localização deve manter o registro dos agentes e suas propriedades a fim de ser capaz de automaticamente criar novos escopos e restringir o acesso aos escopos.

O *contexto* de um agente representa as circunstâncias nas quais um agente se encontra. Em geral, um contexto inclui toda a informação sobre o ambiente de um agente que é relevante para sua atividade. O contexto de um agente em CAMA consiste nas seguintes partes: (i) conexões de estados para as localizações; (ii) nomes, tipos e estados de todos os escopos visíveis nas localizações; (iii) o estado de escopos nos quais o agente está participando atualmente, inclusive as tuplas contidas nestes escopos. Um conjunto de todas as localizações define a estrutura global de contexto do agente. Este contexto muda quando um agente migra de uma localização para outra.

Em CAMA, agentes se comunicam uns com os outros através de um construtor especial do espaço de coordenação chamado *escopo*. Um agente pode cooperar somente com agentes que participam nos mesmos escopos. O uso de escopos provê uma maneira de estruturar atividades de agente pelo arranjo destes em grupos de acordo com suas intenções. Escopos também permitem que a configuração da comunicação de agentes se adapte aos requisitos de seus grupos.

2.4.3. CARISMA

Context-Aware Reflective mIddleware System for Mobile Applications (CARISMA) é um *middleware* para computação móvel que explora a característica de reflexão a fim de permitir a construção de aplicações móveis sensíveis ao contexto e adaptativas (Capra et al. 2003). Este *middleware* oferece primitivas que descrevem como as mudanças de contexto devem ser tratadas através de políticas estabelecidas para os serviços existentes.

De acordo com o modelo proposto em Capra et al. (2001), CARISMA é responsável por manter uma representação válida do contexto de execução através de interações diretas com o sistema de rede subjacente. Neste modelo, entende-se por contexto qualquer coisa que possa influenciar o comportamento de uma aplicação, desde recursos em um dispositivo, tais como memória, energia da bateria, tamanho da tela, poder de processamento, a recursos fora do dispositivo físico, tais como largura de banda, conexão ou localização e até mesmo recursos definidos pela aplicação, como atividade e perfil do usuário.

CARISMA também é responsável por prover uma interface reflexiva a partir da qual seu comportamento pode ser modificado de acordo com a adaptação desejada pela aplicação. Aplicações podem requerer que alguns serviços sejam executados de diversas maneiras, através de diferentes políticas, quando requisitados em contextos diferentes. Por exemplo, uma aplicação de envio de mensagens pode enviar mensagens de texto quando a largura de banda é alta, enquanto troca mensagens comprimidas quando a largura de banda é baixa.

O foco do desenvolvimento de CARISMA foi a configuração das políticas e o tratamento de conflitos que ocorrem quando mais de uma política se aplica a um contexto. O *middleware* trata uma situação de conflito distribuído quando várias aplicações interagem entre si por um mesmo serviço e os perfis das aplicações selecionam políticas diferentes para o mesmo serviço.

Como mostra a Figura 4, a arquitetura de CARISMA possui quatro componentes principais (Capra, 2003):

(i) *Core*, fornece as funcionalidades básicas, como o suporte à comunicação assíncrona e descoberta de serviços;

(ii) *Context Management*, responsável por interagir com sensores físicos e monitorar as mudanças de contexto;

(iii) *Core Services*, serviços responsáveis por responder às requisições de qualidade de serviços definidas no nível de aplicação; e

(iv) *Application Model*, define um *framework* padrão para criar e executar aplicações sensíveis ao contexto sobre o modelo de *middleware*.

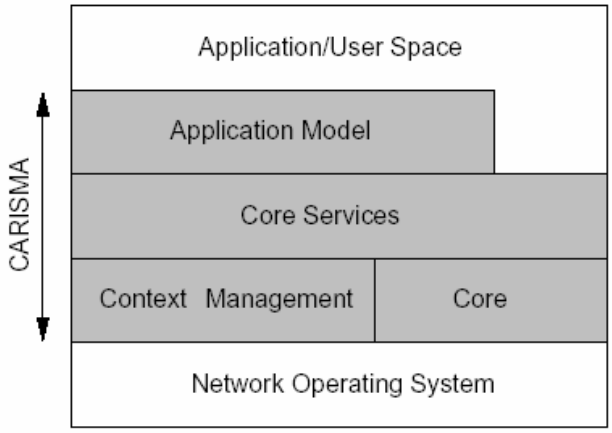

Figura 4. Arquitetura CARISMA (Capra et al., 2003)

2.5. Análise dos *Middlewares*

De modo geral, os *middlewares* existentes na literatura não são capazes de permitir o desenvolvimento de aplicações sensíveis ao contexto robustas. Com exceção do *framework* CAMA, os outros *middlewares* não provêem qualquer suporte ao tratamento de exceções, sem o qual as aplicações não são capazes de satisfazer os requisitos de robustez esperados.

Mesmo no caso de CAMA, a sensibilidade ao contexto não é devidamente explorada para auxiliar as tarefas relacionadas ao tratamento de exceções. Não se faz menção a requisitos importantes do tratamento de exceções sensível ao contexto: a propagação de erros não considera mudanças contextuais que ocorrem constantemente nos sistemas, as atividades de recuperação de erros e a estratégia de tratamento de exceções não são selecionadas de acordo com as informações contextuais e exceções não são caracterizadas a partir do contexto.

Portanto, faz-se necessário o desenvolvimento de uma abordagem de tratamento de exceções que considere as mudanças de contexto de aplicações móveis, ou seja, uma abordagem de tratamento de exceções sensível ao contexto.