1
Introdução

A Computação Móvel desempenha um papel crescente no desenvolvimento de sistemas de software em função do rápido avanço das novas tecnologias de comunicação utilizadas nos dispositivos portáteis (Chen & Kotz, 2000; Varshney & Vetter, 2000; Gaddah, & Kunz, 2003). Com a Computação Móvel, cresce também a popularidade de tecnologias relacionadas à sensibilidade ao contexto, as quais possibilitam aos sistemas monitorar e utilizar informações relevantes que provêm do ambiente ou dos usuários dos sistemas (Mitchell, 2002; Coutaz et al, 2005). Torna-se assim possível que usuários explorem as possíveis oportunidades de colaboração enquanto se movimentam através dos ambientes.

Por outro lado, o uso de paradigmas adequados é essencial na tarefa de construção de sistemas de software, devido à própria complexidade das aplicações emergentes (Jennings, 1999; Lind, 2000). Por exemplo, o desenvolvimento de aplicações sensíveis ao contexto, ao explorar a especificação de propriedades complexas de Engenharia de Software como o tratamento de exceções, se constitui como uma tarefa não-trivial (Iliasov & Romanovsky, 2005; Tripathi et al, 2005). Tais propriedades poderiam ser mais facilmente tratadas se as abstrações e os mecanismos dos paradigmas utilizados no desenvolvimento destas aplicações pudessem satisfazer também requisitos de qualidade, como modularidade e reusabilidade.

1.1.
Problema

Nos últimos anos, deu-se um grande avanço na busca de novas tecnologias envolvendo áreas como as de comunicação celular, de redes locais sem fio e de serviços via satélite, a fim de viabilizar o acesso e a utilização de recursos computacionais de forma remota. De fato, a popularização de dispositivos portáteis como *notebooks*, PDAs e *smartphones*, bem como a larga

utilização de redes sem fio em *campi* de universidades, aeroportos e residências, são resultado direto desta busca por novas tecnologias de acesso remoto.

Em particular, as pesquisas na área de Computação Móvel (Chen, 2000; Varshney & Vetter, 2000; Gaddah, & Kunz, 2003) têm como principal objetivo o desenvolvimento de técnicas para a integração dos sistemas portáteis com aplicações existentes nos mais diversos contextos. Nesta área de pesquisa, o conceito de computação ubíqua, proposto por Weiser (1991), tem apontado para a necessidade do desenvolvimento de aplicações capazes de responder a mudanças dinâmicas em um contexto através de mínima interferência humana. Tais aplicações são chamadas "aplicações sensíveis ao contexto" (ASCs) (Mitchell, 2002; Coutaz et al, 2005).

ASCs podem ser desenvolvidas de várias maneiras (Baldauf & Dustdar, 2004; Chen, 2004), mas atualmente há uma tendência para o uso de *middlewares* que dão suporte ao gerenciamento de informações de contextos nos mais diferentes domínios (Henricksen & Indulska 2005; Viterbo et al, 2006). Alguns deles também fornecem interfaces de programação de aplicações (APIs) para o desenvolvimento de ASCs (Chen, 2004; Khedr & Karmouch, 2004; Rubinsztejn et al, 2004). Entretanto, mesmo com todas as facilidades disponíveis, a construção de ASCs tem se mostrado como tarefa não-trivial.

De fato, características relacionadas especificamente à sensibilidade ao contexto, como abertura[1], comunicação assíncrona e falta de mecanismos modulares na propagação de informações de contexto, constituem alguns dos principais fatores de complexidade de ASCs (Iliasov & Romanovsky, 2005; Tripathi et al, 2005). Como o foco deste trabalho trata de questões gerais de Engenharia de Software encontradas em ASCs que decorrem diretamente do uso das APIs de *middlewares*, apresentamos, a seguir, uma breve análise de conceitos presentes em tais *middlewares* que, se especificados a partir de um paradigma de desenvolvimento de software mais adequado, poderiam mensuravelmente facilitar o tratamento de propriedades de ASCs, como o tratamento de exceções.

[1] Do inglês, *openness*.

1.2. Limitações das Soluções Existentes

Middlewares utilizados no desenvolvimento de ASCs são, em geral, baseados em alguns dos seguintes elementos: (1) *sensores* são responsáveis por coletar informações de contexto; (2) *provedores* são encarregados de popular as ASCs com as informações de contexto, geradas a partir dos sensores ou de outras informações; (3) *consumidores* registram interesses por informações de contexto ou por eventos baseados em contexto; (4) *brokers* são responsáveis pelo registro de interesses e a disseminação das informações de contexto.

MoCA (Rubinsztejn et al, 2004) é um exemplo de *middleware publish-subscribe*, em que cada aplicação possui servidores em uma rede fixa e clientes nos dispositivos móveis. O *Context Information Service* (CIS) é o serviço de MoCA que processa as informações de contexto, recebe registros de interesse nestas informações e, além disso, gera e envia eventos aos interessados quando ocorrem mudanças em um dado dispositivo. Além do CIS, o *Location Inference Service* (LIS) infere a localização aproximada de um dispositivo móvel nas ASCs. Além destes serviços, MoCA disponibiliza APIs para o desenvolvimento de ASCs, como *Client* API e *Server* API. A Figura 1 ilustra a arquitetura de MoCA.

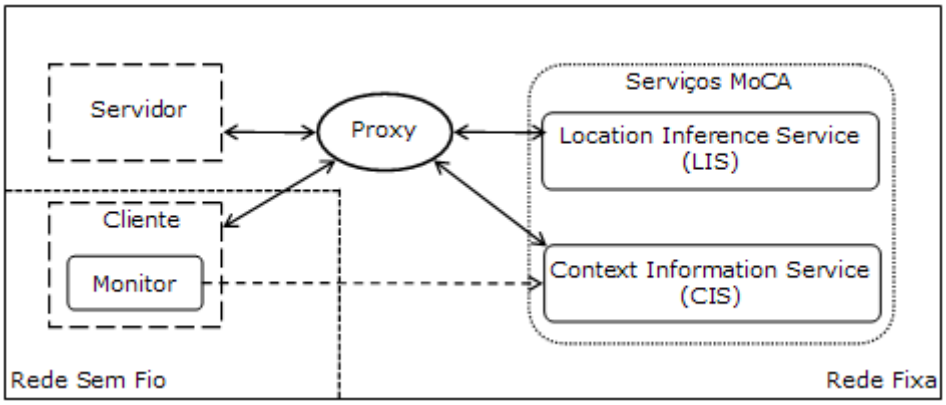

Figura 1. Dependências entre as APIs de MoCA

Outro exemplo de arquitetura utilizada no desenvolvimento de ASCs é CoBrA (*Context Broker Architecture*), proposta por Chen (2004). Esta arquitetura (Figura 2) possui uma base de conhecimento que armazena informações de contexto representadas em forma de triplas em *Resource Description Format* e uma máquina de inferência, o *Context Reasoning Engine*, que permite determinar novas informações de contexto a partir de dedução lógica baseada na semântica

do OWL e regras de dedução. CoBRA também disponibiliza uma API para criação de ASCs, além de um módulo responsável pelo gerenciamento da política de privacidade, o *Module for Privacy Protection*, que determina o compartilhamento de informações de contexto por usuários.

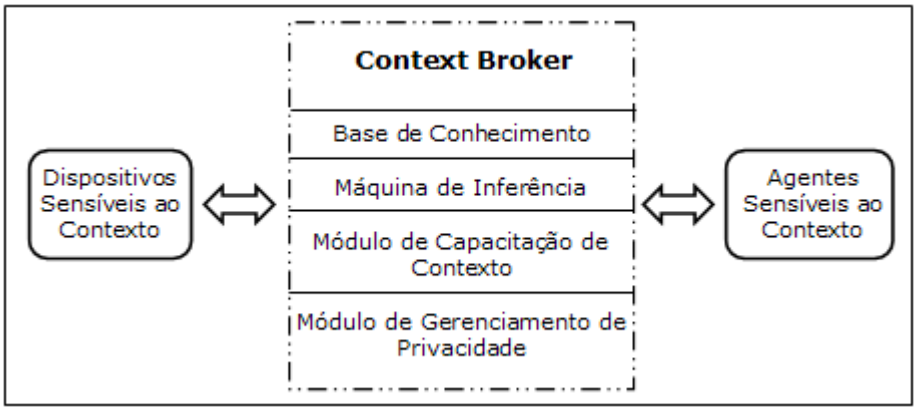

Figura 2. Arquitetura de CoBrA

ACAI (*Agent-Based Context-Aware Infrastructure*) é também utilizada no desenvolvimento de ASCs (Khedr & Karmouch, 2004), implementada segundo uma estrutura de camadas (Figura 3): na *camada de percepção*, as informações de contexto são capturadas e coletadas; a *camada de serviços de contexto* é responsável pela descoberta dos serviços disponíveis no ambiente, pelo armazenamento e consumo das informações por outros serviços e pela dedução de informações que não estão disponíveis diretamente a partir da camada de percepção; a *camada de aplicação* funciona como uma API que permite que as informações de contexto sejam negociadas entre provedores e consumidores.

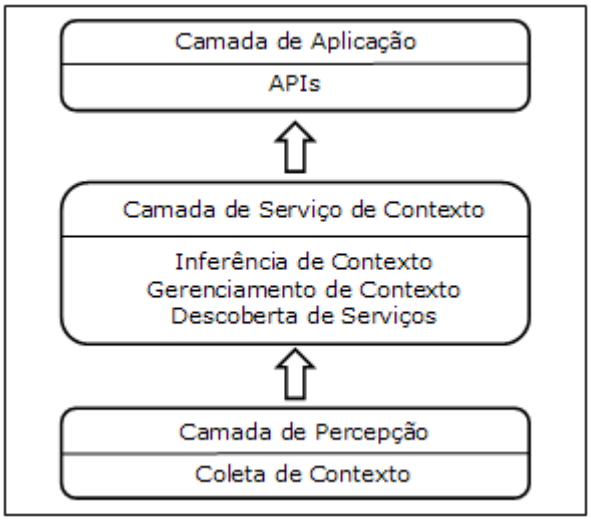

Figura 3. Arquitetura ACAI

Da descrição anterior, podemos afirmar que, do ponto de vista da Engenharia de Software, as abstrações e os mecanismos providos pelos *middlewares* de ASCs estão quase que estritamente relacionados aos requisitos específicos de sensibilidade ao contexto. Em conseqüência, apesar do uso das APIs disponíveis, os engenheiros de software ainda precisam tratar de várias questões não diretamente relacionadas ao domínio das ASCs, que poderiam ser mais facilmente tratadas dependendo das abstrações disponíveis para tal.

Por exemplo, "contexto" (Seção 2.1) é um dos conceitos fundamentais em ASCs. Segundo Dey e Abowd (1999), contexto é qualquer informação utilizada para caracterizar a situação de uma pessoa, lugar ou objeto relevante para a interação entre um usuário e uma aplicação sensível ao contexto. Entretanto, a representação e manipulação de um contexto podem variar sensivelmente a depender do *middleware* sendo utilizado. Existem várias possibilidades para a representação de contextos (Strang et al, 2004): pares atributo-valor, objetos, gráficos, baseada em esquemas, lógica e ontologias.

Ocorre que, um contexto poderia ser mais facilmente tratado pelos engenheiros de software se pudesse ser interpretado apenas como um *objeto* cujos atributos armazenam dados sobre o ambiente, independentemente de sua representação interna em um *middleware* específico. Por outro lado, outros conceitos, apesar de recorrentes, são independentes de *middlewares* e, por suas propriedades complexas, poderiam ser mais adequadamente tratados se fossem considerados como objetos "complexos" com características de autonomia, interação e adaptação, pois agregam questões relacionadas à manipulação de *threads*, troca de mensagens assíncronas e implementação de *listeners* de eventos.

Além disso, há diferenças significativas nos serviços e na forma como esses serviços são expostos para o desenvolvedores de ASCs. Isso traz como conseqüência mais complexidade para o engenheiro de software, que tem que aprender a utilizar diferentes APIs de *middlewares*. Por exemplo, apenas 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. Por outro lado, muitas vezes o papel do sensor se confunde com o do provedor de informação e os provedores também podem ser usados para atuar como consumidores, sobretudo quando implementam a inferência de contexto.

Tendo em vista esta complexidade decorrente do uso direto das APIs de *middlewares*, algumas pesquisas em andamento têm por objetivo verificar a usabilidade de novas abstrações para o desenvolvimento de ASCs (Harroud et al, 2004). De fato, é relevante que os engenheiros de software sejam providos de abstrações e mecanismos de decomposição mais próximos do domínio das ASCs, do que daqueles especificamente relacionados à sensibilidade ao contexto.

1.3. Solução Proposta

Os paradigmas de desenvolvimento de software são ferramentas básicas na Engenharia de Software, inclusive na tarefa de implementação das APIs disponibilizadas pelos *middlewares* para desenvolvimento de ASCs. Eles possuem a função de fornecer abstrações e mecanismos de decomposição que facilitem o tratamento da complexidade dos sistemas em geral.

Particularmente, a *Engenharia de Software para Sistemas Multi-Agentes* (ESSMA) tem se apresentado como paradigma de desenvolvimento promissor para o desenvolvimento de aplicações distribuídas, abertas e extensíveis (Jennings, 1999; Lind, 2000; Garcia, 2004; Silva, 2004; Garcia, 2005).

De fato, *agente de software* é uma técnica que pode auxiliar no desenvolvimento de ASCs, complementando os serviços oferecidos por APIs existentes, pois suas propriedades básicas são propriedades que freqüentemente encontramos em ASCs (Cacho et al, 2006b; Damasceno et al, 2006; LAC).

Por exemplo, a *interação* se refere à capacidade de um agente em se comunicar com usuários e outros agentes através de alguma linguagem e, sendo assim, podemos considerar que tal característica seja comum em ASCs, pois dispositivos móveis são capazes de se comunicar em um ambiente distribuído.

Mais ainda, como, por definição, as ASCs se adaptam baseadas em informações coletadas do estado corrente de seus usuários, do ambiente físico e de outros sistemas presentes, podemos afirmar que tal característica pode também ser "pensada" através da *adaptação* de agentes, que está relacionada às respostas dadas por agentes a eventos notificados pelo ambiente onde estão inseridos.

Além disso, como a *autonomia* diz respeito ao funcionamento de um agente de software sem a intervenção direta de usuários, podemos também afirmar que

ela também freqüentemente emerge em ASCs de tempo real, como é o caso da aplicação de *Health Care* em Cacho et al (2006b).

Outro exemplo é a abertura inerente a sistemas multi-agentes encontrada comumente em ASCs, pois tais aplicações são consideradas como abertas, devido à mobilidade dos dispositivos envolvidos e, sendo assim, agentes de software heterogêneos e previamente desconhecidos podem ser usados para vir a interagir com elementos já presentes nas aplicações (Paes, 2005).

O requisito de propagação modular de informações de contexto também pode ser satisfatoriamente contemplado pelo uso de ESSMA, tendo em vista que as abstrações e mecanismos propostos por este novo paradigma permitem definir um modelo simplificado de ASCs enfatizando apenas os detalhes mais importantes, enquanto permite ignorar outros (Garcia, 2004; Silva, 2004).

Em resumo, partindo das características básicas de sistemas multi-agentes, é possível propor soluções para o desenvolvimento de ASCs que facilitem a satisfação de requisitos comuns nestas aplicações. Neste caso, o desenvolvimento de ASCs é realizado a partir de conceitos mais próximos do domínio das aplicações do que daqueles relacionados à sensibilidade ao contexto.

Como este trabalho busca explorar a usabilidade de ESSMA para o desenvolvimento de ASCs e não propor uma solução alternativa ao uso de *middlewares*, optamos por utilizar aplicações de MoCA (Rubinsztejn et al, 2004) em nossos estudos, por duas razões principais: apesar da documentação, de aplicações e funcionalidades escassas, é, dentre as que procuramos, a mais disponível e também aquela que oferece melhor suporte ao programador.

Além disso, MoCA possui todas as características e funcionalidades geralmente presentes em arquiteturas *publish-subscribe*, o que faz deste *middleware* um ótimo ponto de partida para a demonstração da usabilidade de ESSMA em ASCs.

1.4. Objetivos

Este trabalho tem por objetivo geral analisar, comparar e avaliar estudos de caso envolvendo o uso de ESSMA no desenvolvimento de ASCs utilizando *middlwares publish-subscribe*. Requisitos comuns nestas aplicações, como

abertura, assincronismo na troca de mensagens, adaptação a eventos de contexto, modularidade de propagação das informações de contexto e o comportamento autônomo de elementos internos a ASCs, são tratados através das novas abstrações e mecanismos propostos por ESSMA.

São objetivos específicos:

1. o levantamento de requisitos de ASCs especificamente relacionados à sensibilidade ao contexto e a análise dos conceitos potenciais de ESSMA que podem ser utilizados para a satisfação de tais requisitos;
2. a reengenharia de ASCs usando as abstrações e mecanismos de decomposição propostos por ESSMA;
3. a proposta de um *framework* para facilitar o desenvolvimento de ASCs em conjunto com outras ferramentas existentes, como *middlewares* para ASCs e plataformas de agentes;
4. uma análise quantitativa e qualitativa de ASCs codificadas *com* e *sem* o uso de ESSMA;
5. a demonstração dos benefícios do uso de abstrações e técnicas de ESSMA em ASCs, através da medição de atributos internos de software, como acoplamento, coesão e tamanho.

1.5. Organização do Texto

Este trabalho está organizado como a seguir. No Capítulo 2, é apresentado o levantamento de requisitos de ASCs especificamente relacionados à sensibilidade ao contexto que potencialmente podem ser tratados a partir de novos paradigmas de desenvolvimento de software. No Capítulo 3, é apresentada uma revisão da literatura sobre ESSMA, o paradigma usado neste trabalho para a reengenharia das ASCs *Virtual Lines* e *Wireless Marketing Service*. No Capítulo 4, é apresentado o *framework* CAAF, que possibilita o reuso em ASCs a partir da integração de soluções existentes. No Capítulo 5, são apresentadas as avaliações qualitativa e quantitativa do uso de ESSMA em ASCs. Finalmente, são apresentadas as conclusões do trabalho, além de propostas para trabalhos futuros.