4
Questões de Middleware

Durante o planejamento da aplicação proposta, questões relacionadas à sua propriedade colaborativa, à edição de quadros de uma apresentação e ao acesso a informações de contexto, tiveram que ser resolvidas para facilitar o desenvolvimento da mesma. O principal objetivo foi escolher ou desenvolver componentes independentes que pudessem ser re-utilizados em quaisquer outros projetos.

4.1.
Colaboração

A colaboração entre os usuários da aplicação proposta envolve detalhes de nível de rede como: a maneira como a conexão será realizada; o envio e recebimento de mensagens; e o tratamento de possíveis falhas durante a comunicação. Esses detalhes requerem um grande esforço por parte dos desenvolvedores de uma aplicação colaborativa, já que ao invés de estarem concentrados no desenvolvimento da aplicação em si, os desenvolvedores perdem um tempo razoável no desenvolvimento desta comunicação.

A escolha de um *middleware* que encapsule as funcionalidades da comunicação é uma forma natural de facilitar o desenvolvimento de uma aplicação colaborativa. Outras vantagens são a confiabilidade do *middleware*, que pode ter sido utilizado em outros projetos com sucesso e a agilidade no processo de desenvolvimento da aplicação, visto que todas as funcionalidades necessárias à comunicação estarão contidas no *middleware*.

4.1.1.
Motivação da Escolha do ConferenceXP

O ConferenceXP [ConferenceXP, 2006] foi escolhido como *middleware* de colaboração da aplicação proposta devido a facilidade de desenvolver aplicações colaborativas com o mesmo. Desenvolvedores podem criar aplicações

colaborativas ou adaptar aplicações já existentes utilizando as *APIs* e o conjunto de classes básicas do ConferenceXP. Este *middleware* facilita a tarefa dos desenvolvedores, pois trata questões relacionadas à comunicação de aplicaçãoes que devem colaborar entre si. A Figura 12 exibe a arquitetura de uma aplicação colaborativa que utiliza o ConferenceXP, dividindo a mesma em dois níveis: nível de rede, onde o ConferenceXP realiza a comunicação; e o nível de aplicação, onde são realizadas as funcionalidades principais da aplicação.

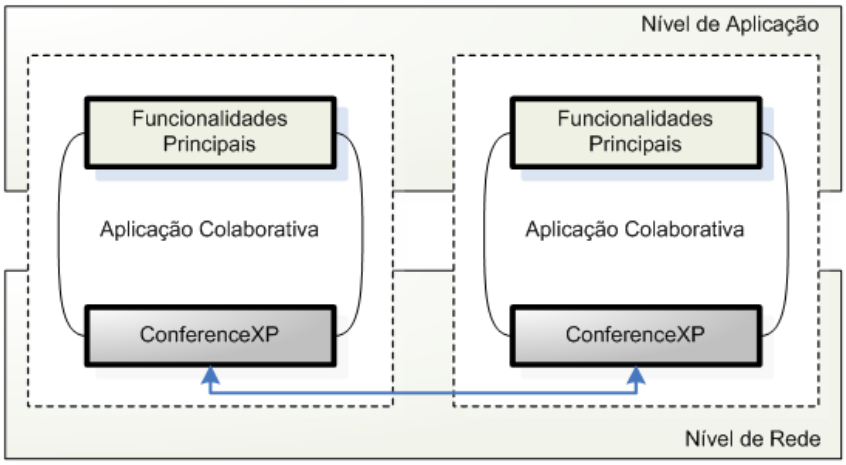

Figura 12 – Aplicação colaborativa utilizando o ConferenceXP

O ConferenceXP foi desenvolvido para a plataforma .NET Framework [NET, 2007], na linguagem C# [C#, 2007], mesma linguagem de desenvolvimento da aplicação proposta, o que facilita a integração e o aprendizado do mesmo. Este *middleware* emprega uma arquitetura *peer-to-peer* ao utilizar *multicast*, não havendo necessidade de um servidor que receba e repasse todas as mensagens, o que facilita a implantação de qualquer aplicação. O uso de *multicast* ao invés de *unicast* garante um menor consumo de recursos de rede, como visto na Figura 13 [Multicast, 2007].

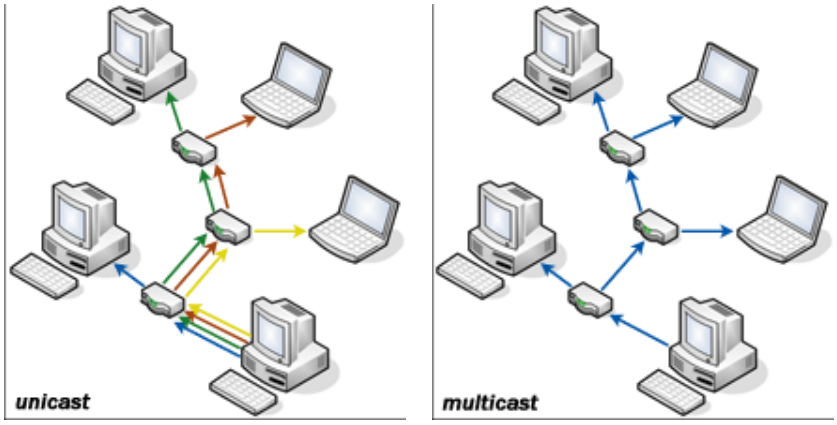

Figura 13 - Diferenças ente o *unicast* e o *multicast* [Multicast, 2007]

O Classroom Presenter (vide seção Classroom Presenter2.1) foi desenvolvido utilizando o *middleware* ConferenceXP, o que comprova sua eficiência e confiabilidade no desenvolvimento de aplicações colaborativas.

4.1.2. Arquitetura do ConferenceXP

O ConferenceXP é dividido em quatro camadas lógicas: Network Transport, ConferenceAPI, ConfereceXP Capability e ConferenceXP Application. A Figura 14 ilustra esta arquitetura do ConferenceXP.

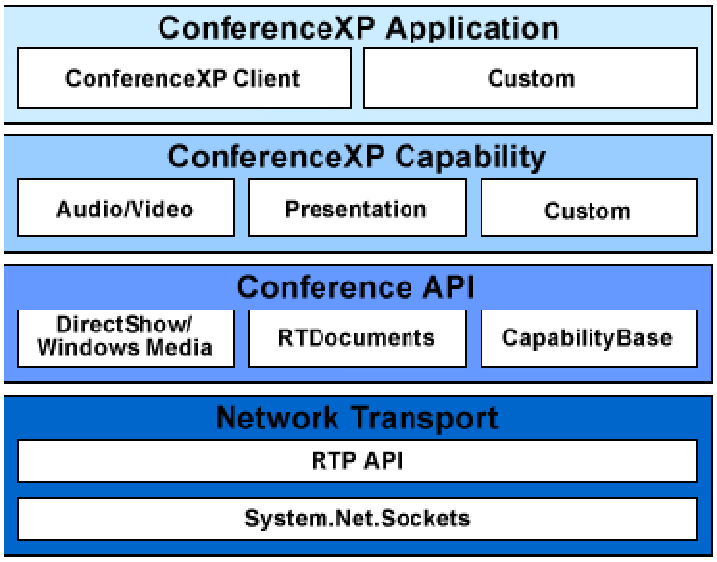

Figura 14 - A arquitetura do ConferenceXP [ConferenceXP, 2006]

A camada **Network Transport** oferece uma implementação do protocolo RTP (Real-Time Transport Protocol) [RTP, 2007], baseado na implementação dos Windows Sockets. Este protocolo é um padrão da IETF (Internet Engineering Task Force) [IETF, 2007] para transmissão de áudio e vídeo em redes *peer-to-peer*, e foi desenvolvido para ambientes onde é necessário um baixo nível de latência. Para prevenir perda de dados, o ConferenceXP implementa algoritmos FEC (Forward Error Correction).

Na camada **ConferenceAPI**, os desenvolvedores podem criar aplicações colaborativas ou *capabilities*, sem se preocupar com aspectos de rede. A classe *CapabilityBase* encapsula a funcionalidade requerida de outras partes da camada de conferência e serve como base para criação de novas *capabilities*, que são componentes que oferecem funcionalidades às aplicações do ConferenceXP. Com a API RTDocument, aplicações e *capabilities* utilizam um protocolo para transferência de documentos e *inks*.

Na camada **ConferenceXP Capability**, originalmente existem duas *capabilities*: a de apresentação (*Presentation*), que oferece suporte à apresentações PowerPoint® e ao uso de *ink*, e a de áudio e vídeo, utilizada para conferências. Nesta camada estão as *capabilities* customizadas desenvolvidas para aplicações colaborativas, que junto com a camada **ConferenceXP Application,** oferecem interfaces aos usuários.

4.1.3. Limitações e o .NET Compact Framework

Para entender melhor as limitações do ConferenceXP, é necessário primeiro distinguir as diferenças entre o .NET Framework e o .NET Compact Framework [CFNET, 2007a]. O .NET Compact Framework (.NET CF) é um subconjunto do .NET Framework que oferece interoperabilidade com o sistema operacional Windows Mobile de um *handheld* como um *palmtop* ou um *smartphone* [CFNET, 2007b]. Ao comparar as duas plataformas de desenvolvimento, a arquitetura do .NET CF é bastante limitada, e não possui diversas funcionalidades, como o suporte nativo a serialização de objetos. Este recurso é essencial para aplicativos de colaboração, onde objetos são enviados e recebidos através da rede [CFNET, 2007c].

Outra limitação é o tratamento dado à coleta de *ink*. O ConferenceXP utiliza um componente desenvolvido para *tablet pcs,* chamado **Microsof.Ink**. Este componente é utilizado no desenvolvimento de aplicações que utilizem a caneta do *tablet pc* para captura de desenhos e reconhecimento de textos. O Microsoft.Ink é restrito ao .NET Framework, e, portanto, não pode ser utilizado em uma aplicação que deva executar em dispositivos mais limitados como *palmtops*.

Estas limitações do .NET CF em relação ao .NET Framework refletem as limitações do ConferenceXP, que não foi desenvolvido para ser utilizado em dispositivos que utilizam o .NET CF.

4.1.4. Adaptações Necessárias

O ConferenceXP faz uso de vários métodos e classes que só existem no .NET Framework, não possuindo iguais correspondentes no .NET CF. Logo, uma série de adaptações foi necessária para que fosse possível sua utilização em ambas as plataformas, atendendo assim os requisitos da aplicação proposta.

É importante notar que todas as aplicações desenvolvidas para o .NET CF que utilizem somente as funcionalidades do próprio *framework*, ou seja, não utilizem funcionalidades nativas dos dispositivos portáteis utilizados, podem ser executadas no .NET Framework. Assim, o trabalho de adaptação do ConferenceXP foi realizado da seguinte maneira: a partir do código-fonte das *APIs*, foram realizadas alterações para compilação e execução no .NET CF. Algumas classes tiveram que ser implementadas para preservar as funcionalidades originais do *middleware*, como: *DicitonaryBase*, classe abstrata que serve como base para coleções de pares chave/valor; e *SynchronizedQueue*, implementação de uma fila sincronizada. Várias alterações em métodos utilizados no código destas *APIs* também foram necessárias durante o trabalho de adaptação.

Também foram utilizadas as bibliotecas do OpenNETCF [OpenNETCF, 2007], para funcionalidades como o acesso a arquivos e configuração e métodos de reflexão. O OpenNETCF é um projeto apoiado pela Microsoft®, e tem como objetivo desenvolver funcionalidades que não existem no .NET CF.

4.1.4.1. Serialização de Objetos

Como o .NET CF não possui suporte nativo a serialização de objetos, foi utilizado um projeto independente chamado CompactFormatter [CompactFormatter, 2007], que é um formatador genérico para o .NET CF capaz de serializar a maioria dos objetos utilizando reflexão, permitindo redefinir os algoritmos de serialização se necessário. A interface disponibilizada por este componente é bastante similar à do *BinaryFormatter* e do *SOAPFormatter*, formatadores utilizados comumente no .NET Framework. É válido mencionar que a serialização entre as diferentes plataformas também é complexa devido ao fato de que um mesmo objeto pode possuir diferentes implementações em cada plataforma. Por exemplo, a classe *Hashtable* possui atributos diferentes na versão para .NET Framework em relação à versão para .NET CF. Para a maioria das classes que se encaixam neste perfil, o CompactFormatter utiliza algoritmos que provêm uma solução automática.

A Figura 15 exibe a comunicação entre aplicações em diferentes plataformas. Toda serialização de objetos será feita utilizando o CompactFormatter, mesmo quando ambas as aplicações estiverem sendo executadas no .NET Framework.

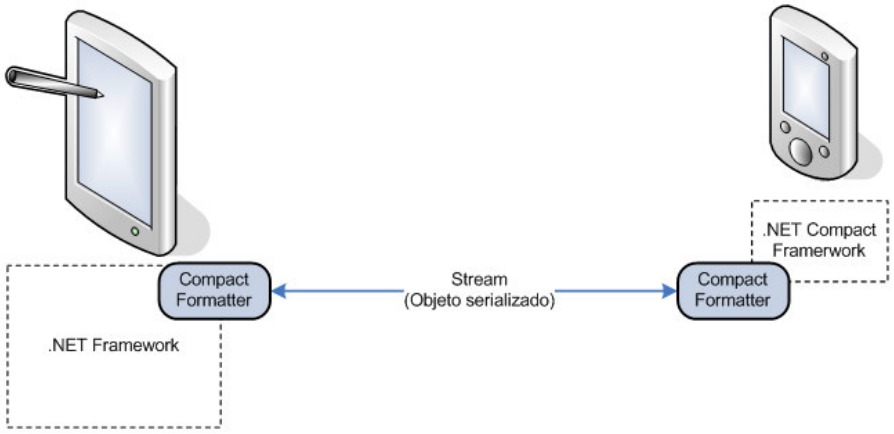

Figura 15 - A serialização de objetos utilizando o CompactFormatter

4.1.4.2. Componente de Edição

O ConferenceXP utiliza o componente Microsof.Ink para a captura de desenhos feitos com a caneta do *tablet pc*. Este mesmo componente é utilizado pelo Classroom Presenter para a edição de quadros, e, portanto não pode ser usado para o desenvolvimento da aplicação proposta. Logo, foi necessário desenvolver um componente que capturasse desenhos realizados em qualquer dispositivo, seja este um *notebook,* um *tablet pc* ou um *handheld,* e oferecesse a adição de textos.

O componente desenvolvido foi chamado de **LAC.Contribs**, e substituiu todas as referências ao componente Microsof.Ink no código adaptado do ConferenceXP. Apesar de possuir uma qualidade inferior em relação ao componente originalmente utilizado, o desenho da *ink* é satisfatório, mesmo quando visualizado em diferentes dispositivos. A Figura 16 exibe um exemplo de uma imagem desenhada em um *palmtop*, e sua visualização em um *desktop*.

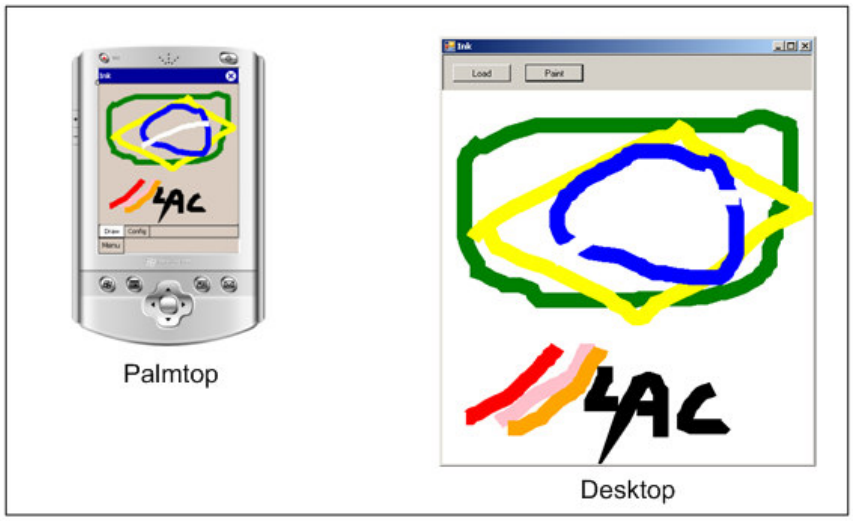

Figura 16 - Desenho feito em um *palmtop* e visualizado em um *desktop*

O LAC.Contribs foi desenvolvido de maneira independente, podendo ser utilizado por quaisquer aplicações que necessitem capturar desenhos ou adicionar textos em ambas as plataformas .NET. Para que uma aplicação possa fazer uso deste componente, somente é necessário que o controle visual definido como área de desenho herde da classe *ComponentBaseControl*. A Figura 17 exibe o diagrama de classes do componente LAC.Contribs.

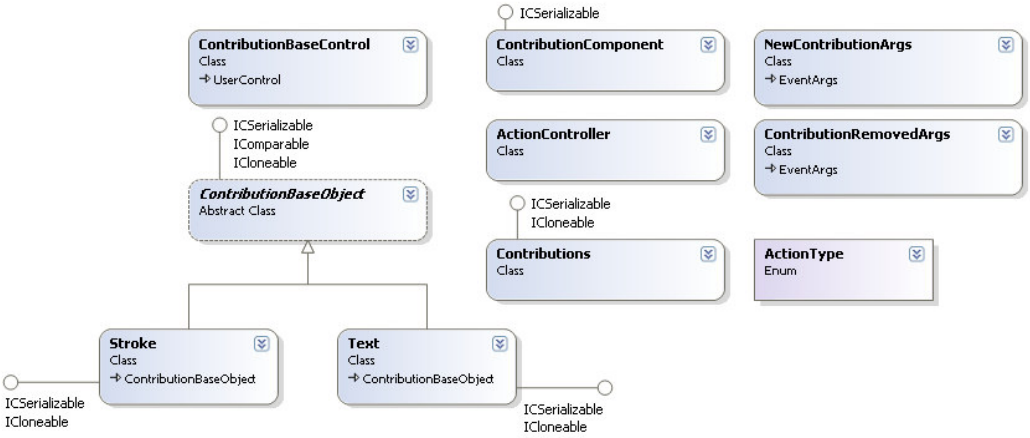

Figura 17 - O diagrama de classes do componente LAC.Contribs

Após a ativação do componente, a partir de um atributo do tipo ActionType, é possível escolher o modo de captura: *ink* ou texto. A *ink* é desenhada a partir das coordenadas de captura da caneta ou do *mouse*, e é possível formatar sua cor e largura. Para o texto, o usuário escolhe a fonte, a cor e o tamanho e clica no ponto onde deseja inserir o texto. Então, uma janela é visualizada e solicita ao usuário o que deve ser escrito. O texto será desenhado no controle visual, como visto na Figura 18.

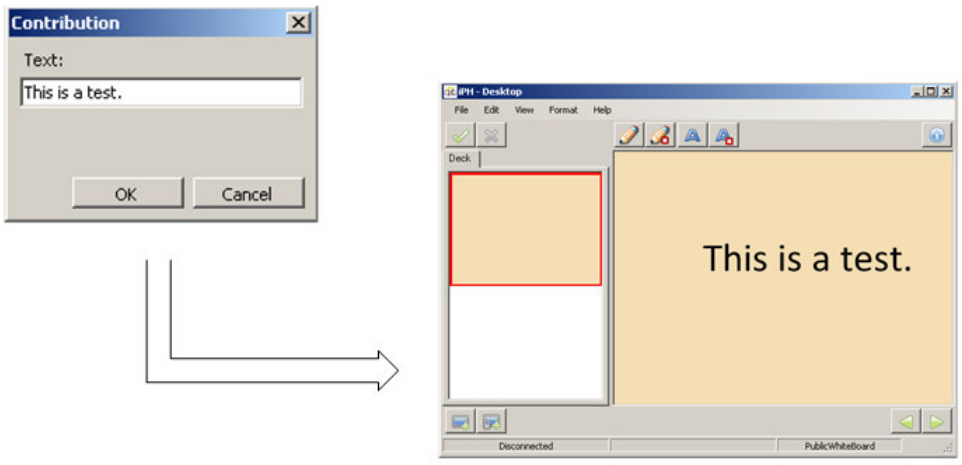

Figura 18 - Inserindo um texto com o componente LAC.Contribs

Com o LAC.Contribs, é possível também apagar os desenhos realizados e os textos inseridos. Ao escolher este modo de borracha, basta que o usuário clique em uma linha desenhada ou em um texto e os mesmos são apagados.

Uma linha desenhada é um objeto da classe *Stroke,* e um texto inserido é um objeto da classe *Text*. Ambas as classes derivam de uma classe abstrata chamada

ContributionBaseObject, e são serializáveis pelo componente CompactFormatter. A classe *ContributionBaseObject* possui atributos de identificação únicos e marcadores de tempo. Os atributos de identificação devem ser únicos, pois para uma aplicação como a proposta neste trabalho, é ser necessário saber, por exemplo, qual das contribuições foi apagada. Assim, a aplicação pode enviar aos demais participantes comandos para exclusão da determinada contribuição. Os marcadores de tempo são atribuídos no momento em que cada objeto é criado, e são necessários para a ordem do desenho dos objetos. Isto garante que não haverá sobreposição incorreta entre os mesmos.

O LAC.Contribs oferece também suporte a eventos disparados pelas ações de criação e remoção de objetos, sejam eles desenhos ou textos. Uma aplicação talvez necessite saber quando um desenho foi realizado, e qual o objeto resultante da ação. Por exemplo, a aplicação proposta necessita saber quando um evento deste tipo ocorreu, pois é necessário enviar aos demais participantes as novas contribuições realizadas pelo mestre.

4.1.5. CompactConferenceXP

Ao adaptar as *APIs* do ConferenceXP para execução também no .NET CF, foram utilizados diversos componentes como o OpenNETCF, o CompactFormatter e o LAC.Contribs, além de várias alterações necessárias. O resultado destas adaptações é o conjunto de *APIs* chamado de **CompactConferenceXP**. Qualquer aplicação que necessite de colaboração entre as diferentes plataformas pode utilizar estas *APIs* para este fim. É importante ressaltar que outras funcionalidades do ConferenceXP não mencionadas neste trabalho como o *Venue Service* e o *Archive Service*, além da aplicação cliente do ConferenceXP não foram adaptadas para utilizar o CompactConferenceXP.

4.2. Informações de Contexto

Como a aplicação proposta deve ser sensível a contexto, esta deve ter acesso a informações como localidade e propriedades físicas dos dispositivos como energia restante da bateria, memória disponível, qualidade da conexão, entre

outras. Com estas informações, a aplicação poderá tomar ações automaticamente, sem a necessidade de interação direta com o usuário. Para o acesso a estas informações, foi utilizado um *middleware* chamado MoCA (Mobile Collaboration Architecture) [Sacramento, 2004], que oferece recursos para o desenvolvimento de aplicações sensíveis a contexto para computação móvel.

4.2.1. MoCA

Através de um conjunto de *APIs*, a MoCA provê serviços eficientes para a coleta, armazenamento e acesso a informações de contexto referentes a dispositivos móveis, e inferência e gerenciamento de informações sobre a localização geográfica de dispositivos. Desenvolvida na linguagem Java [Java, 2007], a MoCA não assume que a aplicação deva ser implementada de acordo com qualquer arquitetura específica, e seus serviços podem ser utilizados como base para o desenvolvimento de uma grande variedade de aplicações.

O monitor é o programa que executa em cada um dos dispositivos móveis, e é responsável por toda a coleta de dados a respeito do estado destes dispositivos. Todas estas informações são enviadas para um serviço da MoCA chamado CIS (Context Information Service). O CIS recebe, armazena e processa as informações de contexto enviadas por um conjunto de monitores. Este serviço também pode receber requisições de notificações de aplicações interessadas em estados específicos de determinados dispositivos. Quando um determinado dispositivo atinge determinado estado, o CIS dispara eventos notificando cada uma das aplicações interessadas.

A localização de um dispositivo é inferida pelo LIS (Location Inference Service) comparando os níveis dos sinais de rádio-freqüência recebidos dos pontos de acesso coletados pelo CIS, aos níveis previamente coletados em determinadas localidades [Nascimento, 2006].

A Figura 19 ilustra a arquitetura da MoCA, e exibe a comunicação entre as aplicações clientes que executam paralelamente os monitores, o CIS, o LIS, e os demais serviços da MoCA.

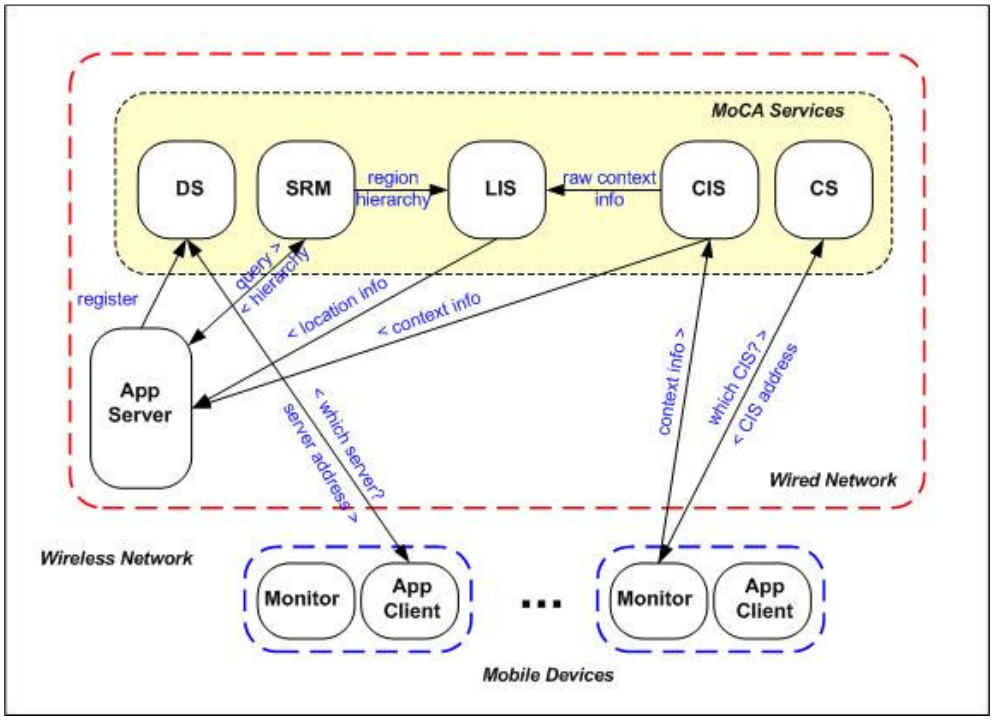

Figura 19 - A arquitetura da MoCA [Moca, 2007]

4.2.2. MoCA/WS

O MoCA/WS (MoCA/*Web Service*) [Malcher, 2006] é um *web service* que foi desenvolvido para que aplicações desenvolvidas em linguagens não-Java, como C++, Visual C#, Visual Basic, entre outras, possam acessar e utilizar os serviços de contexto oferecidos pela MoCA. Este *web service* atua como um cliente da MoCA, e através do envio e recebimento de mensagens SOAP (Simple Object Access Protocol) [SOAP, 2007] em requisições HTTP, qualquer aplicação pode acessar informações de contexto provenientes da MoCA.

O MoCA/WS atua como um *proxy* e oferece interface similar às próprias *APIs* da MoCA para aplicações Java. Quando uma aplicação cliente envia uma requisição ao MoCA/WS, este repassa a requisição à MoCA, que por sua vez retorna a resposta ao *web service.* A Figura 20 exibe a comunicação entre uma aplicação cliente, o MoCA/WS e a MoCA.

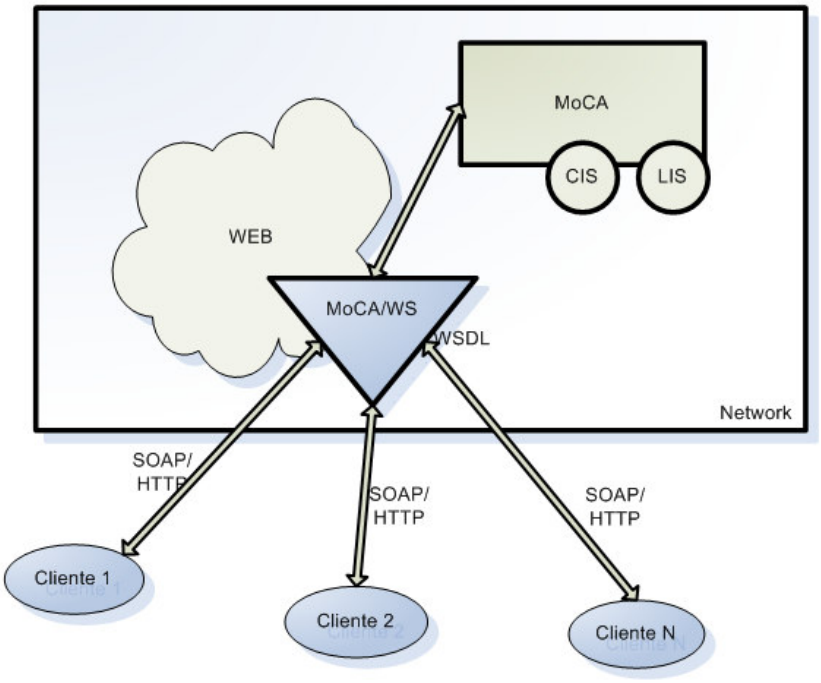

Figura 20 – MoCA x MoCA/WS x aplicações clientes [Malcher, 2006]

Como a aplicação proposta foi desenvolvida em Visual C#, o acesso às informações de contexto da MoCA foi realizado através do MoCA/WS.