Suporte de Middleware Auto Adaptável para Aplicações Móveis e Distribuídas

Adalberto Ribeiro Sampaio Junior[1], Fábio Moreira Costa[2]
Instituto de Informática - UFG, 74001-970, Brasil
{adalberto.comp,fmc.ufg}@gmail.com

PALAVRAS-CHAVE: Interoperabilidade, *Middleware*, Mobilidade, Tolerância a falhas.

1. Introdução

Com os recentes avanços surgindo na computação e comunicação, as pessoas estão passando a conhecer e usar um paradigma que até pouco tempo atrás era desconhecido de muitos, a computação móvel e ubíqua ou pervasiva (SUN, 2005).

Tecnologias ubíquas permitem que as pessoas experimentem situações nas quais os dispositivos computacionais estão presentes em todos os lugares ao seu redor, se comunicando entre si e com a Internet, de uma forma transparente para os usuários e provendo serviços de acordo com o contexto em que se encontram (ADELSTEIN, 2004).

Computação móvel é uma os ingredientes chave para a computação ubíqua. Em sua forma mais avançada, a computação ubíqua permite que as pessoas interajam com os sistemas como se estivessem conversando com alguém, não percebendo que estão dando comandos a um computador (WEISER, 1991).

Mas para que aplicações ubíquas sejam construídas de uma forma mais simples, é necessário que exista uma infraestrutura que trate os principais desafios da ubiquidade, dentre eles a mobilidade e a interoperabilidade das aplicações. Para isso, sistemas de *middleware* se fazem necessários (GADDAH, 2003).

O uso de middleware permite a criação de um ambiente distribuído onde os problemas oriundos da distribuição do sistema ficam ocultos para o programador. Preocupações típicas de caráter não-funcional, tais como localização de serviços e

[1] Aluno do Bacharelado em Ciência da Computação - INF-UFG
[2] Orientador - INF-UFG

recursos, uso de diferentes linguagens de programação, interoperabilidade e replicação, tornam-se transparentes, deixando o programador livre para lidar com a funcionalidade das aplicações.

Porém, ambientes onde existe a presença de dispositivos móveis são caracterizados por possuírem um alto grau de dinamismo e, para amenizar as dificuldades que podem ser encontradas nesses ambientes um *middleware* adaptativo se faz necessário (MELO, 2009).

Neste trabalho, será utilizado um middleware adaptativo conhecido por Meta-ORB (COSTA, 2001), que será modificado para dar suporte à mobilidade e interoperabilidade das aplicações e dispositivos computacionais que o utilizarem. Isto constituirá a base para uma subsequente extensão do middleware para o suporte a aplicações ubíquas.

O restante deste relatório está estruturado como se segue. A Seção 2 descreve o ambiente que motivou as alterações a serem feitas no MetaORB.NET. A Seção 3 nos mostra uma visão geral da plataforma de middleware MetaORB.NET e seus principais componentes que foram modificados neste trabalho. A Seção 4 discute os requisitos que conduziram às novas funcionalidades do middleware, bem como a forma como elas foram implementadas. Finalmente, a Seção 5 discute as contribuições deste trabalho e potenciais trabalhos futuros.

2. Visão Geral da Arquitetura

O MetaORB.NET é uma implementação em linguagem C#, sobre a plataforma .NET, da arquitetura Meta-ORB de middleware adaptativo. Uma discussão mais detalhada da implementação dos recursos básicos do middleware pode ser encontrada em (PROVENSI, 2006). Assim, nesta seção serão abordados apenas os aspectos básicos da implementação do middleware.

Na implementação atual, a plataforma MetaORB.NET oferece as seguintes funcionalidades :

- Núcleo – fornece suporte para o modelo de programação da plataforma Meta-ORB, com construções que representam, em tempo de execução, as entidades de primeira classe do modelo, como componentes, interfaces e *bindings*. Além disso, contém a infraestrutura básica do middleware e a

implementação padrão de componentes especializados, como as fábricas de componentes e de *bindings*;

- Serviços Remotos – Serviços globais, como o repositório de tipos e o serviço de nomes, que, para manter a interoperabilidade com a plataforma MetaORB4Java, foram implementados em Java e acessados através de serviços Web;
- Meta-nível – Implementação dos mecanismos reflexivos da plataforma. Esses mecanismos fornecem a funcionalidade adaptativa da plataforma, mas sua descrição foge ao escopo do presente trabalho. Sua especificação pode ser encontrada em (COSTA, 2002).

2.1. Modelo de Programação

A plataforma MetaORB.NET segue um modelo de programação baseado em componentes (PROVENSI, 2009), adotado principalmente por facilitar a construção de configurações e sua posterior reconfiguração. Com isso, a plataforma oferece maior facilidade para a construção de configurações e reflexão para adaptação dinâmica.

Existem três elementos básicos que formam qualquer sistema baseado na plataforma Meta-ORB, incluindo o próprio núcleo da plataforma, sendo estes os Componentes, suas Interfaces e os *Bindings*. Uma descrição desses elementos é mostrada a seguir.

2.1.1. Componentes

Componentes são as unidades da plataforma que possuem funcionalidades (das aplicações ou do próprio middleware) encapsuladas, podendo ser acessados por meio de uma ou mais interfaces. Embora o modelo de programação da plataforma permita a construção tanto de componentes primitivos quanto compostos (componentes construídos na forma de configurações de componentes internos), a implementação atual da versão MetaORB.NET permite apenas a construção de componentes primitivos. Os componentes primitivos encapsulam de forma atômica as funcionalidades das aplicações e do middleware e

representam as unidades básicas de um sistema baseado na plataforma MetaORB.NET.

2.1.2. Interfaces

As interfaces são os únicos pontos de acesso à funcionalidade dos componentes. Uma aplicação baseada na plataforma Meta-ORB.NET pode possuir interfaces de três tipos: operacional, de fluxo e de sinal.

As interfaces operacionais são utilizadas para dar suporte a interações onde as operações providas por uma interface de um componente servidor podem ser utilizadas por um componente cliente por meio de uma interface compatível. Para tanto, deve haver uma ligação explícita entre a interface do componente servidor (na qual as operações são declaradas como 'providas') e a interface do componente cliente (onde as mesmas operações são declaradas como 'requeridas'). Isto permite a realização de interações do tipo requisição-e-resposta, na forma de chamadas remotas de métodos.

As interfaces de fluxo são usadas para interações entre os componentes de aplicações multimídia. Elas permitem a comunicação de fluxos de dados multimídia (por exemplo, áudio e vídeo) entre componentes produtores e consumidores.

Por fim, interfaces de sinal definem interações primitivas e de sentido único entre as interfaces de componentes. Esse tipo de interação não oferecem nenhum tipo de retorno ou garantia de chegada do sinal ao componente solicitante e é normalmente utilizado para notificar a ocorrência de eventos.

2.1.3. *Bindings*

O último elemento do modelo de programação é o objeto de *binding*. Esses objetos são responsáveis por intermediar a interação entre componentes remotos e podem ser tanto primitivos quanto compostos.

Bindings primitivos são uma forma de encapsular o protocolo de transporte subjacente, fornecendo uma interface para comunicação remota em conformidade com o modelo de programação da plataforma, isto é, por meio do uso de interfaces operacionais, de fluxo e de sinal. *Bindings* primitivos são normalmente usados apenas como parte da configuração de *bindings* compostos.

Bindings compostos, por sua vez, são usados para representar estruturas de comunicação mais complexas e são construídos por meio da agregação de bindings primitivos e componentes que fornecem funcionalidades de comunicação mais elaboradas. Externamente, contudo, *bindings* compostos e *bindings* primitivos possuem as mesmas características, sendo acessíveis por meio de interfaces que seguem o mesmo modelo de programação. Em (PROVENSI, 2009) é apresentada uma descrição mais detalhada desse tipo de *binding*.

Bindings compostos também são chamados de *bindings explícitos*, significando que sua implementação interna pode ser visualizada e manipulada com a finalidade de adaptação, por exemplo, para controle de qualidade de serviço.

2.2. Cápsula

Os serviços básicos de suporte ao *middleware* são providos por um objeto especial do núcleo da plataforma, chamado de Cápsula. Uma cápsula, instanciada em um dispositivo, representa a unidade básica de localização para todos os objetos criados dentro de um ambiente distribuído. As cápsulas, durante sua inicialização cria os componentes necessários para os serviços básicos oferecidos pelo middleware, como os componentes de suporte a comunicação e as fábricas de componentes e *bindings*.

As cápsulas também possuem um sistema de *caching* para diminuir o *overhead* da busca de referências das interfaces no servidor de nomes (descrito abaixo). Uma vez que uma dada interface é recuperada no servidor de nomes, ela é adicionada à *cache* local na cápsula. Dessa forma, caso seja feita uma nova busca por essa mesma interface, ela é recuperada a partir da *cache* local, sem a necessidade de acesso remoto ao servidor de nomes.

A cápsula provê também suporte para dois tipos primitivos de interação entre componentes: *bindings* locais e *bindings* implícitos. *Bindings* locais não possuem estrutura interna e são responsáveis por conectar duas interfaces locais (localizadas na mesma cápsula) compatíveis, ou seja, os serviços requeridos de uma interface são providos pela outra interface. *Bindings* implícitos, por outro lado, destinam-se à comunicação cliente-servidor com componentes remotos e não possuem estrutura interna explicitada (e, portanto, não são adaptáveis), o que os torna mais leves e apropriados para realizar interações ocasionais entre

componentes que não requerem algum tipo de controle de qualidade de serviço. Como o nome indica, um *binding* implícito não precisa ser estabelecido a priori. A comunicação pode ser simplesmente utilizando a referência da interface remota para chamar suas operações. *Bindings* implícitos servem unicamente para o acesso a interfaces operacionais.

2.3. Servidor de Nomes

Na plataforma Meta-ORB, o acesso a interfaces remotas é feito por meio de referências, conhecidas com *IRefs*. Para isso, a plataforma fornece um serviço que gerencia as referências, o Servidor de Nomes. Sua principal função é armazenar as IRefs e indexá-las usando nomes únicos, definidos no momento da criação dos objetos e suas interfaces.

Sempre que um componente é criado, suas interfaces são registradas pelo serviço de nomes local, que cria uma referência para a interface no servidor de nomes global. Assim, de posse do nome único de uma interface, uma aplicação pode obter por meio de uma consulta ao servidor de nomes, a referência dessa interface, a qual pode ser usada para invocação remota direta de métodos (por meio de um *binding* implícito) ou para estabelecimento de um *binding* explícito entre uma interface local e a interface remota.

2.4. Serviço de Comunicação

O MetaORB.NET possui um sistema de comunicação interno composto por dois componentes que permitem o acesso a operações nas interfaces de componentes remotos por meio de *bindings* implícitos. Esses componentes *CommServer* e *CommClient* foram construídos para servir de base para a invocação remota de métodos.

Cada cápsula do MetaORB.NET possui esses dois componentes, de forma que uma cápsula pode tanto expor interfaces ao sistema, através do *CommServer*, como utilizar interfaces remotas, através do *CommClient*.

3. Suporte a Mobilidade e Interoperabilidade

O suporte a mobilidade e interoperabilidade é fundamental para que aplicações móveis tenham um comportamento aceitável em ambientes heterogêneos (SUN, 2005). Para prover esse suporte, a plataforma MetaORB.NET foi atualizada e um módulo de interoperabilidade foi adicionado, juntamente com um módulo de gerência de mobilidade, ao seu serviço de comunicação. Essas duas extensões representam a contribuição principal do presente trabalho.

3.1. Interoperabilidade

A interoperabilidade entre diferentes implementações da plataforma, notadamente a Meta-ORB.Java e Meta-ORB.NET, é baseada na definição de uma representação externa de dados, de forma que esses dados possam ser comunicados em um formato independente de sua representação interna na plataforma. Em linguagens orientadas a objetos, os dados a serem representados externamente compreendem desde simples tipos primitivos a grafos inteiros de objetos relacionados entre si.

Na nossa implementação, primeiramente foi definido um núcleo de serialização, que permite que os dados, tanto primitivos quanto grafos de objetos (desde que sejam acíclicos), sejam serializados. Esse núcleo de serialização utiliza reflexão computacional para inspecionar os objetos a serem serializados. Isto significa que o desenvolvedor não precisa se preocupar com a implementação interna dos objetos a serem serializados. O núcleo verifica os tipos de dados em tempo de execução e cria uma representação do objeto na forma serializada.

Na implementação do módulo de serialização, o padrão JSON (JSON, 2011) foi escolhido como formato da representação externa dos objetos da plataforma MetaORB.NET. Tal escolha se deve a quatro principais características principais do JSON:

Formato textual Documentos JSON são representados como cadeias de caracteres codificados, i.e., em texto plano. Isto permite a depuração da comunicação entre cápsulas da plataforma;

Estrutura hierárquica Sua característica hierárquica permite uma melhor representação de objetos aninhados;

Compressão Devido à sua natureza textual, o formato JSON é mais fácil de ser entendido, por exemplo, para fins de depuração.

Tamanho final do arquivo serializado Por ter uma sintaxe simples o arquivo gerado em JSON possui um tamanho pequeno em comparação com outros formatos, como XML (XML). Isso contribui para diminuir a quantidade de dados trafegados na rede.

Nesta implementação, os tipos primitivos do *framework* .NET (int, char, string, etc.), são passíveis de serialização além de vetores e das interfaces *Collection* e *Map*.

No caso de objetos, para que um objeto seja serializado, sua classe deve ter uma anotação *Marshallable(true)*.. Dessa forma, tanto os tipos de dados primitivos quanto,objetos complexos que possuam essa anotação são serializáveis. Caso os objetos possuam objetos anotados, e estes possuam referências cíclicas, o módulo de serialização lança uma exceção informando que não é possível realizar serialização neste tipo de objeto.

Além dessa anotação cada atributo do objeto, seja ele um tipo primitivo ou outro objeto, deve conter métodos `get()` e `set()` para que o framework de reflexão do .NET seja capaz de recuperar os dados em tempo de execução. Dessa forma quando o método `serialize()` é chamado, tendo o objeto a ser serializado como parâmetro, um arquivo JSON é criado com a representação completa do objeto.

O Código 2 mostra o documento JSON e o correspondente documento XML (para fins de comparação), ambos gerados a partir da serialização de uma instância da classe `Pessoa`, definida no Código 1.

```
[Marshallable(true)]
    class Pessoa
    {
        public string nome { get; set; }
        public List<string> cursos { get; set; }

        public Pessoa()
        {
            cursos = new List<string>();
            nome = "Adalberto";
            cursos.Add("Ciência da Computação");
            cursos.Add("Sistemas de Informação");
        }
    }
```

Código 1 Classe Pessoa

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<object Pessoa>
      <string>Adalberto</string>
      <collection>
            <string>Ciência da Computação</string>
            <string>Sistemas de Informação</string>
      </collection>
</object>
```

JSON

```
"Pessoa" : {
      "string":"Adalberto"
      "collection": [
            " Ciência da Computação ",
            " Sistemas de Informação "
      ]
}
```

Código 2 Comparação entre a serialização da classe Pessoa em XML e JSON, note que o resultado em JSON é menor que em XML

3.2 Protocolo de Mobilidade

Apesar da evolução das redes sem fio na última década, essa tecnologia traz tanto benefícios quanto limitações às aplicações. Dentre as principais limitações, podemos citar a baixa oferta de banda, concorrência pelo meio de transmissão e a necessidade de *handover* (migração de uma rede para outra em consequência da mobilidade). Este último, em particular, pode ser problemático, uma vez que pode causar a desconexão de aplicações distribuídas e a consequente interrupção de serviços no sistema. Como consequência, dispositivos móveis, podem perder a conexão de rede temporariamente ou enfrentar baixa qualidade no sinal. Além disso, os dispositivos móveis podem ter seu endereço de rede alterado por diversos motivos, incluindo mudança de rede.

Em dispositivos móveis essa troca de endereço pode acontecer com uma frequência bem maior em relação a dispositivos computacionais fixos (PITOURA, 1997). Devido ao fato desse tipo de dispositivo ser capaz de se conectar a diversas tecnologias de rede sem fio (GPRS, 3G, Wi-Fi, etc.), um *handover* entre estas

tecnologias pode acontecer frequentemente (com o objetivo de diminuir alguns dos problemas das redes sem fio, como baixa oferta de banda), o que pode inviabilizar o funcionamento de uma aplicação distribuída caso não exista um tratamento para este comportamento dinâmico.

Com base nessas observações e na aplicação construída em (PROVENSI, 2006) e estendida em (SAMPAIO JUNIOR, 2010), foram colhidos requisitos para melhoria do serviço de comunicação da plataforma para que haja um tratamento da mobilidade levando em consideração o tipo de interface de cada componente da aplicação.

3.2.1. Requisitos dos três tipos de interface

Como mostrado na Seção 2.1.2, cada tipo de interface na plataforma Meta-ORB representam um comportamento particular na interação entre os componentes em um sistema distribuído. Isso faz com que cada tipo de interface necessite de considerações diferentes no tratamento de mobilidade.

Nas interfaces operacionais, os componentes envolvidos atuam como clientes e servidores. Nesse tipo de interação, sempre existe uma resposta para cada requisição, independente desta ser uma exceção ou um valor nulo. Essas interfaces representam um tipo de comunicação mais sensível à falhas. Em aplicações onde o tempo não é um requisito primordial, as requisições podem ser repetidas até que o receptor da requisição esteja disponível ou por um número estabelecido de vezes. Mas em aplicações sensíveis ao tempo de resposta, na ocorrência de falhas, imediatamente algum valor de retorno deve ser provido pelo mecanismo de comunicação para indicar a falha e permitir que alguma decisão de adaptação seja tomada.

As interfaces de sinal representam o tipo de comunicação mais primitivo existente na Meta-ORB. Este tipo de interação é utilizado principalmente em aplicações que necessitam de comunicação em tempo real. Tais aplicações podem necessitar de informações providas pelo mecanismo de comunicação para adaptar-se a mudanças no ambiente, por exemplo, caso a cápsula que hospeda o componente receptor de um sinal não esteja disponível imediatamente.

Em interfaces de fluxo, um componente produtor emite um fluxo de dados a um ou mais componentes consumidores. O propósito dessas interfaces é oferecer suporte para aplicações multimídia na plataforma Meta-ORB. Contudo, essas

interfaces não oferecem um mecanismo que permite à aplicação adaptar-se na ausência de comunicação entre o produtor e os consumidores. Tal mecanismo deve oferecer um meio de adaptação e considerar dois tipos principais de multimídia: multimídia em tempo real e multimídia baseada em *buffer* (LAGO, 2004).

Cada interface de componente no Meta-ORB possui um nome único registrado no servidor de nomes. Esta característica da plataforma afeta os três tipos de interface. Assim, independente do tipo de interação, todos os nomes registrados no servidor devem ser atualizados, após uma ocorrência de mobilidade, quanto à nova localização da cápsula.

Analisando os requisitos de cada tipo de interface, pode-se listar como requisitos de um módulo de gerência de mobilidade os que seguem:

- Garantir a localização correta das entidades da plataforma no servidor de nomes;
- Garantir a manutenção dos *bindings* das interfaces operacionais visando garantir a entrega da resposta de uma operação;
- Oferecer informações acerca da ocorrência de falhas de comunicação entre interfaces de sinal, permitindo a adaptação baseada em tais informações;
- Oferecer um mecanismo de tolerância a falhas que considere diferentes tipos de multimídia distribuída.

3.2.2. Implementação

A implementação do módulo de mobilidade foi feita com base em alterações no serviço de comunicação da plataforma, além de pequenas alterações na cápsula e no serviço de nomes, para que uma aplicação continue acessível após mudar seu endereço de rede.

O serviço de nomes é constituído por dois componentes, o servidor de nomes e o sistema de *caching* das cápsulas. Na implementação atual da plataforma Meta-ORB, os nomes (e respectivas referências de interface) armazenados no sistema de *caching* não possuem tempo de vida. Uma vez recuperados no servidor, eles são mantidos na cache da cápsula indefinidamente. Com isso, a mobilidade dos dispositivos móveis fica comprometida, pois as referências não são atualizadas para apontar para a nova localização dos componentes e suas interfaces.

Para resolver esse problema, foram feitas também alterações no núcleo da plataforma. Foi adicionado um método ao servidor de nomes (`update( )`). Este método é responsável por atualizar todas as referências de entidades de uma cápsula com seu novo endereço de rede. Por fim, em todas as referências de interface foi adicionado um novo atributo com o tempo de vida da referência.

Como resultado, sempre que o endereço de rede de uma cápsula móvel é alterado, um evento é disparado e uma *thread* atualiza o endereço no servidor de nomes através do método `update( )`. Para as demais cápsulas que já possuem uma referência para esta cápsula móvel, apenas quando uma nova requisição for feita a ela que sua referência será atualizada no *caching* local destas cápsulas.

Dessa forma, sempre que um nome é consultado na *cache* de uma cápsula, caso a referência correspondente não esteja expirada ou não exista no *cache*, a cápsula repassa a solicitação ao servidor de nomes. Referências de interface especiais podem atribuir um valor negativo ao seu tempo de validade; dessa forma o servidor de nomes sempre irá testar sua existência quando esta for requisitada.

Como a atualização das referências é feito de acordo com o seu tempo de vida, caso ocorra alguma falha de comunicação enquanto uma referência ainda for válida, para cada tipo de interface será tomada uma atitude pertinente como será mostrado mais adiante.

Como toda interação entre componentes remotos é feita através de referências de interfaces, todos os requisitos de gerência de mobilidade refletem basicamente na implementação de um único método, `invoke ( )` do componente *CommClient*. Esse método é usado internamente na plataforma como parte da implementação do mecanismo de *binding* implícito. Ele é responsável por realizar chamadas de métodos remotos e recebe como argumentos a referência da interface remota, o nome do método a ser chamado e os parâmetros a serem passados ao método.

Para garantir a entrega de resposta em uma operação e informar a ocorrência falhas na comunicação entre as interfaces, um atributo foi adicionado à classe *BaseInterface*. Esse atributo, `monitorLocation` é responsável por informar o módulo de comunicação sobre como se comportar quando uma referência de interface não está atualizada, isto é, quando a cápsula que possui a interface indicada não está disponível. O atributo é do tipo booleano e pode assumir os

valores `true` ou `false`. O uso desse atributo depende do tipo de interface em questão e é explicado a seguir.

Como já mencionado anteriormente, cada tipo de interface funciona de uma maneira diferente. Devido a essa particularidade, para cada tipo de interface uma ação diferente é tomada quando ocorre alguma falha na comunicação entre as cápsulas.

Para interfaces operacionais, quando o atributo `monitorLocation` está configurado como `true` e ocorre alguma falha, a requisição é armazenada e o método `tryConnection( )` é invocado. Esse método registra uma solicitação de localização da cápsula móvel no servidor de nomes. O retorno dessa solicitação é assíncrono e é enviado à cápsula solicitante apenas quando a cápsula móvel atualizar sua localização. Após a chamada desse método, uma *thread* é executada para monitorar a cápsula móvel através de sucessivas tentativas de conexão à sua última localização conhecida. Esta *thread* é executada até que uma das seguintes condições de parada seja alcançada:

- O servidor de nomes responde com a nova localização da cápsula;
- A tentativa de conexão é bem sucedida;
- Um tempo limite pré-estabelecido é atingido.

Caso a última condição de parada seja alcançada, uma exceção é lançada.

Em interfaces de sinal e fluxo, o tratamento dado às falhas de comunicação é diferente, uma vez que sinais e fluxos possuem um comportamento de tempo real, ao contrário das interfaces operacionais.

Nas interfaces de sinal, sempre que ocorre algum erro, uma exceção é lançada. Nenhuma tentativa de restauração da conexão é feita. Essa abordagem é adotada devido ao fato de que sinais geralmente são utilizados em aplicações que necessitam de comunicação em tempo real e a demora na entrega de um sinal, utilizando uma abordagem similar àquela empregada para interfaces operacionais, pode ser tão prejudicial como a falha total na sua entrega. Por exemplo, um sinal pode ser enviado ao sistema de adaptação de voo de um avião não tripulado caso ocorra um atraso para o envio desse sinal graves consequências podem ocorrer.

No caso das interfaces de fluxo, por se tratarem de transporte de dados multimídia (vídeo, áudio, etc.), a perda de alguns pacotes de dados em alguns casos pode não ser tão prejudicial ao bom funcionamento aplicação. Porém pode haver

situações que a perda destes pacotes seja tão prejudicial quanto sinais perdidos, porém ao contrário dos sinais, estes pacotes podem tolerar um certo atraso no seu envio. Com base nessa abordagem as interfaces de fluxo podem ser tratadas de duas maneiras:

1. Quando a perda de alguns pacotes é tolerável na aplicação, o atributo `monitorLocation` recebe o valor false e um contador de perda de pacotes é iniciado. Assim, quando ocorre a tentativa de envio de um pacote e ocorre algum erro, o pacote é descartado, o contador é incrementado e é feita uma nova busca pela interface solicitada com o objetivo de atualizar a referência dessa interface na cápsula solicitante. Essa sequência de passos (tentativa de envio de um pacote, atualização do contador e busca pela interface) é feita até que o contador atinja um limite pré-estabelecido. Caso este limite seja atingido uma exceção é lançada.
2. Quando a perda de pacotes não é tolerável, mas pode haver algum atraso na entrega, os pacotes são *bufferizados,* o atributo `monitorLocation` recebe o valor `true` e o tratamento de cada pacote é feito da mesma forma que as operações de interfaces operacionais. Este tratamento se faz útil, por exemplo, quando um arquivo está sendo enviado pela rede. É necessário que exista um tratamento para que não ocorra perda de pacote de dados evitando assim que o arquivo se torne corrompido ou inválido.

Com estas extensões introduzidas na plataforma, é possível o desenvolvimento de aplicações móveis distribuídas com uma garantia de confiabilidade, já que o Meta-ORB oferece um sistema de manutenção dinâmica das referências remotas. Isso proporciona que dispositivos móveis façam *handover* sem que ocorra inconsistência no registro de sua interfaces no servidor de nomes, e torna possível que cápsulas remotas acessem cápsulas móveis sem problemas.

4. Considerações Gerais

A principal contribuição deste trabalho se concentra no estudo e aperfeiçoamento do Meta-ORB como *middleware* reflexivo adaptativo para prover serviços configuráveis em tempo de execução.

Muitos trabalhos estão sendo feitos nessa área (CETINA, 2009; GEORGAS, 2009), os quais, assim como no Meta-ORB, utilizam modelos para representação da

plataforma para permitir sua configuração e adaptação de acordo com as necessidades do ambiente.

Essa abordagem na construção baseada em modelos permite que a plataforma seja alterada sem a necessidade de parar seu funcionamento (BLAIR). Da mesma forma como em linguagens reflexivas, isso permite que a configuração interna plataforma seja inspecionada em tempo de execução para alterar sua implementação interna, adicionando, alterando ou removendo componentes, de forma a satisfazer requisitos que por algum motivo não puderam ser colhidos na fase de projeto.

Além disso, o uso de modelos como uma referência da plataforma permite que a adaptação seja autonômica (CETINA, 2009). Como a plataforma possui, através do modelo, conhecimento de sua estrutura interna, é possível que ela, sem intervenção humana, encontre uma configuração apropriada para uma determinada situação e ambiente de execução.

Dessa forma o uso do Meta-ORB, assim como de outras plataformas do gênero, é indicado em ambientes nos quais não é uma opção parar o sistema para atualização. Além disso, plataformas de middleware auto-adaptativas, são necessárias para construção de aplicações nas quais se faz necessário que uma adaptação consistente e eficaz seja feita sem a intervenção humana (GEORGAS, 2009).

5. Trabalhos Futuros

Com o desenvolvimento do módulo de serialização e do módulo de mobilidade foram identificados alguns trabalhos futuros que eliminarão limitações e auxiliarão no desenvolvimento de aplicações.

Mesmo com as alterações feitas na plataforma o tratamento de mobilidade ainda não é tão natural. O frequente monitoramento das cápsulas aumenta o overhead na execução da plataforma, além aumentar o uso da rede para checagem da localização das cápsulas. Isto pode ser prejudicial ao ambiente de execução, principalmente ao se tratar de um ambiente móvel. Pará resolver este problema é proposto a criação de uma nova versão do serviço de comunicação usando o estilo publish/subscribe (EUGSTER, 2003) ao invés de invocação remota de métodos. Usando essa abordagem, o tratamento de mobilidade se tornará mais natural e simples para a plataforma devido às vantagens providas por esse padrão

de requisições e respostas assíncronas. Com isso diminuirá o overhead na execução, e o uso da rede para checagem e manutenção da confiabilidade entre diferentes cápsulas.

Apesar das duas versões da plataforma possuírem as mesmas funcionalidades e conseguirem se comunicar sem problemas, ainda existem alguns detalhes na parte adaptativa que as tornam diferentes. Sendo assim, faz-se necessário evoluir a parte adaptativa do MetaORB4J para que este se torne equivalente ao MetaORB.NET, já que este último possui um melhor suporte a adaptação.

Além disso, o desenvolvimento e atualização da documentação da plataforma devem ser feitos para facilitar alterações do middleware, assim como o desenvolvimento de aplicações. Não existe uma documentação para direcionar como aplicações são construídas e isto se faz necessário para que o uso do Meta-ORB seja mais difundido.

6. Conclusão

Com o desenvolvimento destes módulos o Meta-ORB se torna um *middleware* de propósito geral para computação móvel permitindo que aplicações móveis distribuídas sejam desenvolvidas de uma maneira simples, eliminando a dificuldade de se preocupar com o tratamento da mobilidade e interoperabilidade existente que são alguns dos principais problemas no contexto de computação móvel.

O módulo de serialização de objetos apresenta uma estrutura que faz uso de reflexão computacional. Desta forma, não se faz necessário definir como os objetos serão serializados, sendo todo o processo de serialização automatizado pelo módulo, facilitando seu uso na plataforma. Além disso, com este módulo se torna possível que aplicações sendo executadas sobre o MetaORB.NET possam se comunicar com aplicações sobre a outra implementação em Java, MetaORB4J, de forma transparente já que ambas podem fazer uso dessa representação externa de dados definida pelo módulo.

O módulo de mobilidade cria um método de tolerância a falhas que fornece informações sobre a execução das interações para os níveis superiores da plataforma. Isso permite a adaptação das aplicações na ocorrência de falhas de comunicação causadas pela mobilidade de cápsulas localizadas em dispositivos

móveis. Ainda, este módulo permite a manutenção de *bindings* mesmo quando a conexão entre uma cápsula qualquer e uma cápsula móvel é interrompida.

7. Referências

[ADELSTEIN, 2004] ADELSTEIN, F. et al., **Fundamentals of Mobile and Pervasive Computing**, New York: McGraw-Hill Professional, 2004.

[BLAIR, 2009] BLAIR, G.; BENCOMO, N.; FRANCE , R. B., **Models@ run.time**, Computer, pp. 22-27, Outubro, 2009.

[CETINA, 2009] CETINA, C. et al**, Autonomic Computing through Reuse of Variability Models at Runtime: The Case of Smart Homes**, Computer, pp. 37-43, Outubro, 2009.

[COSTA, 2001] COSTA, F. M., **Combining Meta-Information Management and Reflection in an Architecture for Configurable and Reconfigurable Middleware**. PhD thesis, Lancaster University, Department of Computing, Lancaster, UK, August 2001.

[COSTA, 2002] COSTA, F. M., **Meta-ORB: A Highly Configurable and Adaptable Reflective Middleware Platform**, XX Simpósio Brasileiro de Redes de Computadores - SBRC 2002, Búzios-RJ-Brazil, 2002.

[EUGSTER, 2003] EUGSTER,P. T. et al., **The Many Faces of Publish/Subscribe**, ACM Computing Surveys, 114-131, 2003.

[GADDAH, 2003] GADDAH, A.; Kuns, T., **A Survey of Middleware Paradigms for Mobile Computing**, Technical Report SCE-03-16, Carleton University, 2003.

[GEORGAS, 2009] GEORGAS, J. C.; HOEK, A.; TAYLOR, R. N., **Using Architectural Models to Manage and Visualize Runtime Adaptation**, Computer, pp. 52-60, Outubro, 2009.

[JSON, 2011] JSON, **JavaScript Object Notation,** disponível em: www.json.org (acessado em 08/06/2011).

[MELO, 2009] MELO, C. E. R., **Um framework de Comunicação e Gerencia de Mobilidade para a Plataforma Meta-Orb, Monografia**, Universidade Federal de Goiás, 2009.

[LAGO, 2004] LAGO, N. P.; KON, F., **Processamento Distribuído de Multimídia em Tempo Real com Baixa Latência.** Master's thesis, Universidade de São Paulo, 2004.

[PROVENSI, 2006] PROVENSI, L. L., **Implementação de uma Infra-Estrutura de Suporte a Binding Explícito na Plataforma MetaORB4Java**, Monografia, Universidade Federal de Goiás, 2006.

[PROVENSI, 2009] PROVENSI, L. L., **Uma plataforma de middleware reflexivo com suporte para auto-adaptação.** Master's thesis, Universidade Federal de Goiás, 2009.

[PITOURA, 1997] PITOURA, E.; SAMARAS, G., **Data Management for Mobile Computing**, Springer, 1997.

[SAMPAIO JUNIOR, 2010] SAMPAIO JUNIOR, A. R.; COSTA, F. M., **Análise e Construção de uma Ferramenta de Apoio ao Ensino Baseada em Tablets PCs e Tinta Digital**, VIII Congresso de Pesquisa, Ensino e Extensão, Goiânia, 2010.

[SUN, 2005] SUN, J.; Sauvola J., **From Mobility Management to Connectivity Management**. In: 10th IEEE Symposium on Computers and Communications, pp. 307-312, 2005.

[WEISER, 1991] WEISER, M., **The Computer of 21st Century**, Scientific American, 265(3): 94-104, Setembro 1991.

[XML, 2011] XML, **Extensible Markup Language (XML),** disponível em: www.w3c.org/XML (acessado em 08/06/2011).

[MORIN, 2009] MORIN, B. et al, **Models@ Run.time to Support Dynamic Adaptation**, Computer, pp. 44-51, Outubro, 2009.