UM ESTUDO COMPARATIVO ENTRE SOLUÇÕES DE *MIDDLEWARES* REFLEXIVOS PARA APLICAÇÕES DE ALTA DINAMICIDADE: ESTADO DA ARTE E DESAFIOS DE PESQUISA

Mário Teixeira Lemes, Victor Hugo Lázaro Lopes, João Ricardo Braga de Paiva
Instituto Federal de Educação, Ciência e Tecnologia Goiás
mariolpu@gmail.com, hullopes@hotmail.com, jricardobpaiva@yahoo.com.br

***Abstract*: The monolithic characteristic of the first middleware platforms makes the self-adaptation impossible, since the middleware is not capable of access and change applications or their own internal structures. The inspection and dynamic adaptation capability may be achieved through the usage of more flexible middleware solutions based on reflexive computing. In this article, we aim to present the main solutions of reflexive middleware that provides the inspection and adaptation of applications with high level of dynamicity, and also identify the main research challenges of construction and usage of this middleware category.**

***Keywords:* Reflection; Inspection; Middleware; Adaptation; dynamicity.**

***Resumo*: A característica monolítica das primeiras plataformas de *middleware* impossibilita a auto-adaptação, uma vez que o *middleware* não é capaz de acessar ou modificar sua própria estrutura interna ou das aplicações. A capacidade de inspeção e adaptação dinâmica pode ser conseguida através do uso de soluções de *middlewares* mais flexíveis, baseados em reflexão computacional. O objetivo deste artigo é apresentar as principais soluções de *middlewares* reflexivos que possibilitam a inspeção e a adaptação de aplicações com alto grau de dinamicidade, bem como identificar os principais desafios de pesquisa na construção e no uso desta categoria de *middleware*.**

***Palavras-chave:* Reflexão; Inspeção; Middleware; Adaptação; Dinamismo.**

I. INTRODUÇÃO

As primeiras soluções de *middleware* foram desenvolvidas com o intuito de facilitar o desenvolvimento de sistemas de *software* que suportavam diversas atividades, tais como a computação científica, a descoberta e a disseminação de informação, sem o devido foco e preocupação em sistemas distribuídos [1].

Os avanços na eletrônica sem fio e nas tecnologias de comunicação permitiram o desenvolvimento de novos sistemas distribuídos, móveis e ubíquos, caracterizados pelo alto grau de dinamismo. A computação ubíqua é um paradigma de interação usuário-computador que tem como objetivo o suporte ao usuário em suas tarefas habituais fora do ambiente usual de trabalho [2]. Os objetos do mundo real são equipados com processadores e sensores embutidos, proporcionando assim novas formas de acesso à informação.

Esse novo ambiente computacional trazido pela computação distribuída, móvel e ubíqua precisa de um suporte de uma nova infraestrutura computacional, constituída por protocolos customizáveis, soluções dinâmicas e políticas adaptadas a esse novo cenário. A Figura 1 mostra esse novo ambiente, recheado de novas formas de comunicação, e que precisa atingir os requisitos e a qualidade de serviço esperados pelos usuários.

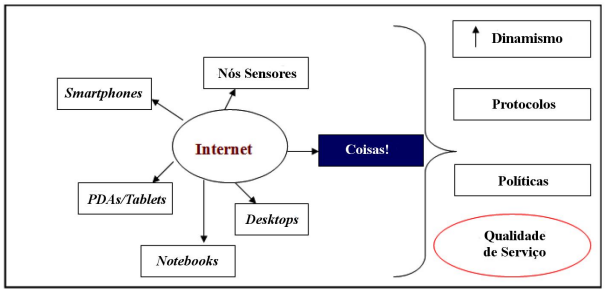

Figura 1 - Influência da computação ubíqua e das aplicações com alto grau de dinamismo

A capacidade de inspeção e adaptação dinâmica, característica marcante em sistemas de *software* ubíquos de alta dinamicidade, pode ser conseguida através do uso de soluções de *middlewares* mais flexíveis, baseados em reflexão computacional.

Um *middleware* reflexivo usa o conceito de refletividade computacional: a aplicação pode acessar algumas partes do estado do sistema e modificá-lo dinamicamente, modificando, então, sua própria estrutura.

O uso da refletividade computacional pode causar resultados inusitados caso seja usada de maneira demasiada, chegando a causar quebra do sistema e incompatibilidade em parte da aplicação.

O *middleware* reflexivo explora o protocolo meta-objeto, combinando as ideias de refletividade computacional e orientação a objetos, sendo dividido

em nível base e nível meta. A ideia do nível base é atingir a funcionalidade das aplicações, enquanto que o meta nível designa coleções de componentes que constituem a arquitetura interna da plataforma do *middleware*. A propriedade reflexiva permite que o comportamento destes objetos seja monitorado, possibilitando mudanças no comportamento do *middleware*.

O *middleware* reflexivo é usado com frequência em aplicações de tempo real, visto que a própria natureza deste tipo de aplicação difere em relação à das já existentes. A flexibilidade introduzida por um *middleware* reflexivo é capaz de suprir as necessidades deste tipo de aplicações, o que não acontece com os *middlewares* tradicionais. Pode-se construir *middlewares* reflexivos através da coleção de componentes colaborativos que podem ser configurados e reconfigurados pela aplicação, com interface imutável e capaz de suportar aplicações de *middlewares* tradicionais.

Como pode ser visto na Figura 2, as principais soluções de *middlewares* desenvolvidas no passado eram baseadas na escala da Internet. Essas soluções são carentes de aspectos dinâmicos. O modelo de *middleware* reflexivo consegue adicionar propriedades dinâmicas requeridas pelas aplicações com alta dinamicidade, como nas aplicações móveis.

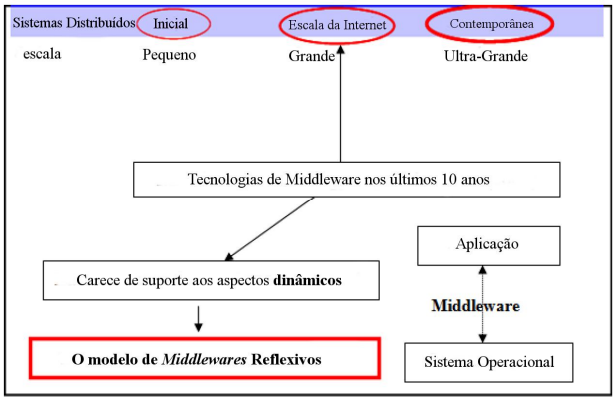

Figura 2 - Principais soluções de *middlewares* carece de aspectos dinâmicos

O objetivo deste artigo é apresentar o estado da arte das principais soluções de *middlewares* reflexivos utilizados em ambientes altamente dinâmicos e analisar as dificuldades e os desafios de pesquisa do uso desta categoria de *middleware* para possibilitar a inspeção e a adaptação das aplicações em tempo de execução.

O estudo comparativo sobre *middlewares* reflexivos se fez necessário dada a não existência de trabalhos que trazem a comparação entre os resultados de estudos e pesquisas de diferentes autores. Dessa forma, reunimos os principais trabalhos sobre *middlewares* reflexivos considerando o impacto dos mesmos, com base em sua quantidade de citações. Para cada um deles foram identificadas as principais características em comum e as peculiaridades de funcionamento, especialmente no que se refere às técnicas de reflexão utilizadas, consequentemente no modo de funcionamento e a qualidade de recursos oferecidos.

O restante deste documento está dividido da seguinte forma. Primeiramente apresentamos conceitos relacionados com *middlewares* reflexivos na Seção II. Em seguida na Seção III, indicamos quais foram os trabalhos selecionados para objeto de comparação neste artigo. Na Seção IV, compactamos os principais conceitos e resultados extraídos dos trabalhos selecionados em um quadro comparativo. Na Seção V apresentamos os principais desafios de pesquisa relacionados com a aplicação e uso de *middlewares* reflexivos. Por fim, concluímos e descrevemos os trabalhos futuros na Seção VI.

II. Conceitos Preliminares

Middleware pode ser definido como uma camada de *software* que contém plataformas distribuídas de interfaces e serviços, e que reside entre a aplicação e o sistema operacional. A interligação de componentes é realizada baseando-se apenas na assinatura de suas interfaces, e não em sua representação [3].

O principal objetivo do *middleware* é facilitar o desenvolvimento, a implantação e o gerenciamento de aplicações distribuídas. A estrutura em camadas da arquitetura de *software* e a posição ocupada pelo *middleware* estão representadas na Figura 3.

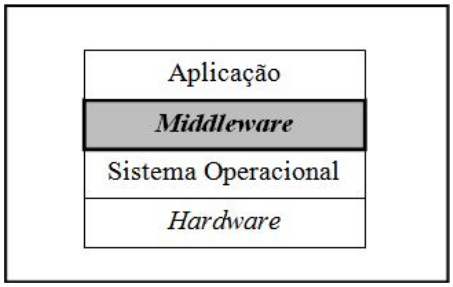

Figura 3 - Estrutura em camadas da arquitetura de software

Uma das funções do *middleware* é esconder os detalhes específicos de camadas inferiores. Essa característica facilita o desenvolvimento de aplicações distribuídas. Os programadores de aplicações distribuídas constroem tais aplicações de modo similar às soluções centralizadas, sem se preocuparem com detalhes internos do sistema operacional ou mesmo com o funcionamento do *middleware*.

Por outro lado, algumas aplicações podem se beneficiar de detalhes específicos providos por camadas inferiores. Imagine uma aplicação de *streaming* multimídia. Se a aplicação é capaz de detectar qual protocolo de transporte está sendo utilizado (TCP ou UDP) e qual é a infraestrutura física da rede (com fio, sem fio, internet de longa distância), a mesma será capaz de escolher as melhores opções para executar o serviço.

Um modelo de *middleware* ideal provê a transparência para as aplicações que assim necessitam e a translucência e um controle de grão fino para as aplicações que se favorecem com a investigação e a análise de camadas inferiores.

As aplicações empregadas em ambientes altamente dinâmicos necessitam que as soluções de *middleware* sejam flexíveis, permitindo que os padrões de interação entre os componentes possam ser adaptados dinamicamente. A facilidade para a configuração dinâmica de módulos da aplicação e gerência de recursos pode ser conseguida através de interfaces para serviços de meta-nível por meio de técnicas de reflexão [1].

O termo reflexão refere-se às capacidades de inspeção e adaptação. De forma mais específica, um sistema reflexivo é aquele que provê a representação do seu próprio comportamento, o que possibilita a inspeção e a adaptação em tempo de execução. A inspeção permite que o estado atual do sistema possa ser observado e a adaptação que o mesmo possa ser alterado. A inspeção e a adaptação são dois aspectos essenciais em sistemas reflexivos.

Um sistema é dito reflexivo quando mantém uma auto-representação causalmente conectada (*Causally Connected Self-Representation* – CCSR) com seu estado e comportamento. A manipulação desta auto-representação é refletida no sistema em si e vice-versa.

Com o uso da auto-representação, a estrutura interna e o comportamento de um sistema computacional podem ser inspecionados e adaptados [4] [5].

O modelo de *middlewares* reflexivos é organizado como uma coleção de diferentes componentes que podem ser configurados e/ou reconfigurados diretamente pela aplicação, em tempo de execução. Essa característica de adaptação pode ser realizada através do uso de meta interfaces. A flexibilidade fornecida pelo uso de *middlewares* adaptáveis e reflexivos está associada ao uso de padrões que utilizam técnicas de interceptação de mensagens, empacotamento e desempacotamento de parâmetros, serialização de objetos e invocação dinâmica de métodos.

As aplicações distribuídas, móveis e ubíquas possuem requisitos altamente dinâmicos. Esta dinamicidade é ainda maior quando se estende o funcionamento dessas aplicações em ambientes de larga escala, com alta taxa de mobilidade e volatilidade, com nenhuma infraestrutura a priori e que precisam conectar uma gama heterogênea de diferentes dispositivos computacionais de forma espontânea e contínua.

Essas características estão presentes no paradigma computacional conhecido como Internet das Coisas (*Internet of Things* – IoT), no qual a junção e a interoperabilidade entre os diversos componentes são um dos principais desafios.

III. Trabalhos Selecionados

Esta seção é responsável por apresentar os principais trabalhos sobre *middlewares* reflexivos presentes na literatura.

A. DynamicTAO [Kon, F. et. al 2000]

DynamicTAO [6] é uma extensão do C++ TAO ORB (Figura 4), desenvolvido na Universidade de Illinois, e que permite a reconfiguração em tempo de execução da máquina interna ORB e das aplicações que são construídas no topo desta máquina interna. As configurações dos componentes representam os relacionamentos dependentes entre os ORBs e os componentes de aplicação [1].

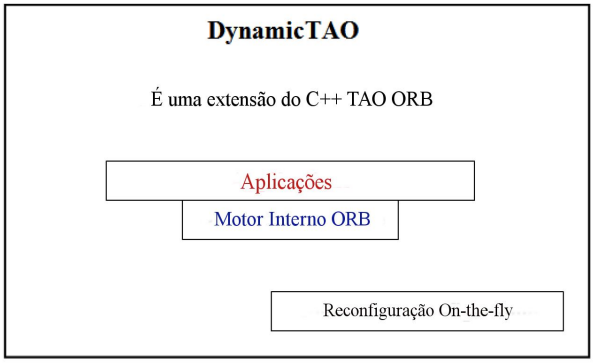

Figura 4 - Estrutura da configuração dos componentes no DynamicTAO

O DynamicTAO usa um arquivo de configuração que especifica as estratégias usadas pelo ORB para implementar aspectos como concorrência, demultiplexação, agendamento e gerenciamento de conexão inter- objeto.

Um configurador de componente carrega as dependências entre certo componente e outro sistema de componentes. Cada processo executando em dynamicTAO ORB contém uma instância de configuração de componente chamada *DomainConfigurator*. Ela é responsável por manter as referências para as instâncias do ORB e para os *servants* executados nesse processo. Estas anotações facilitam a codificação de um componente, pois não é necessário usar arquivos de configuração ou mesmo outras linguagens para especificações do componente. Na Figura 5 tem-se a representação das configurações dos componentes no DynamicTAO.

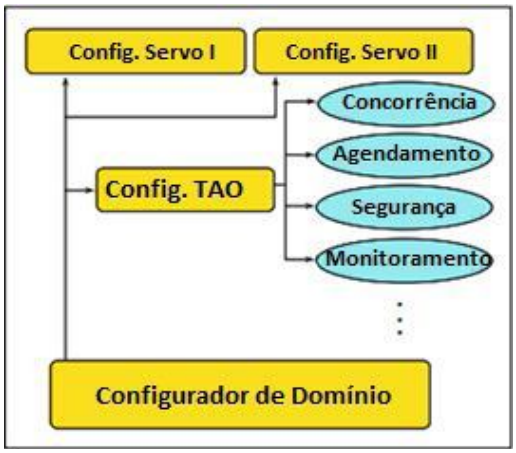

Figura 5 - Estrutura da configuração dos componentes no DynamicTAO

Para permitir a inspeção e a mudança do estado de configuração do ORB, o *middleware* reflexivo DynamicTAO exporta uma meta interface (Figura 6) que permite carregar e descarregar os módulos dentro do sistema em tempo de execução. Uma segunda meta interface também é exportada para agentes móveis. Esses agentes móveis podem ser programados para inspeção e reconfiguração dos ORBs.

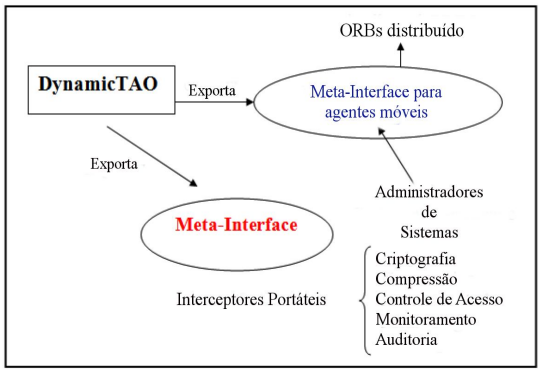

Figura 6 - Interfaces exportadas pelo DynamicTAO

A interposição dinâmica das aplicações no DynamicTAO é possibilitada pelo suporte dado no uso dos interceptadores. Os desenvolvedores podem instalar interceptadores portáteis no lado do cliente ou do servidor e em diferentes níveis de requisição. Note na Figura 7 que para gerenciamento dos recursos, o DynamicTAO utiliza um escalonador dinâmico de tempo real (*Dynamic Soft Real-Time Scheduler* - DSRT).

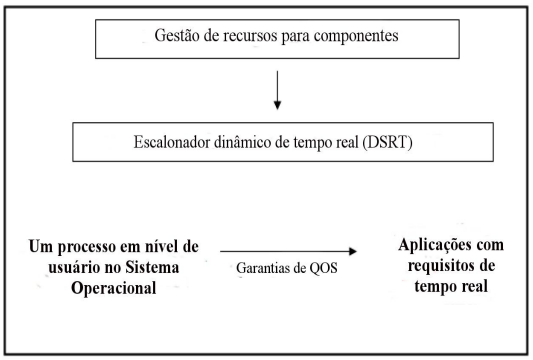

Figura 7 - Escalador Dinâmico de Tempo Real no DynamicTAO

B. Open ORB [Kon, F. et. al 2001]

Desenvolvido na Universidade de Lancaster, o Open ORB [7] teve como principal objetivo o alto poder de configuração/reconfiguração dinâmica para suportar aplicações com requisitos dinâmicos. A reconfiguração dinâmica é conseguida através do uso extensivo de reflexão, com a clara separação entre o nível base e o nível meta [1].

A estrutura do meta espaço no Open ORB é representada na Figura 8. O meta espaço consiste da associação de um componente de nível base a um conjunto privado de componentes de meta nível.

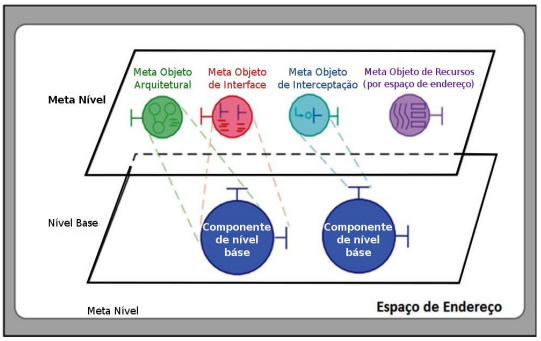

Figura 8 - Associação entre os componentes de nível base e nível meta no Open ORB

Os componentes de nível base implementam serviços usuais de *middleware*, já os componentes de nível meta facilitam a exposição dessas implementações ao programador, o que permite a inspeção e a adaptação. O *middleware* Open ORB

define quatro modelos de meta espaço para reflexão estrutural e reflexão comportamental.

Segundo [1], as características do Open ORB são:

- Arquitetura de meta-níveis bem definidas: os mecanismos convencionais de uma plataforma de *middleware* são implementados no nível base, enquanto que os mecanismos reflexivos da plataforma (que reificam os elementos do nível base) são realizados no meta nível.
- Reflexão orientada a objetos: a meta-representação de um objeto constituinte da plataforma é feita através de meta-objetos, os quais fornecem um protocolo de meta-objetos para acesso às funcionalidades reflexivas. O mesmo modelo de programação, orientado a objetos, é adotado tanto no nível base quanto no meta nível.
- Reflexão procedural: o protocolo de meta-objetos permite acesso (direto) à própria implementação da plataforma, para fins de inspeção e adaptação.
- Meta-objetos individuais: cada objeto constituinte da plataforma pode ser representado, no meta nível, por um (ou mais) meta-objeto (s) exclusivo (s). Entre outras coisas, isto permite limitar os efeitos de reflexão, evitando que os mesmos se propaguem de maneira indesejável.
- Reificação sob demanda: meta-objetos são criados apenas quando necessários. Sendo assim, não é preciso pagar o custo adicional representado pelos mecanismos de reflexão quando os mesmos não forem necessários.

Em relação aos modelos do meta espaço disponíveis no Open ORB, podem-se citar [8]:

- Modelo de Interfaces: permite a reificação da estrutura de um componente (ou objeto) da plataforma em termos de suas interfaces, bem como dos serviços oferecidos por cada interface. O protocolo de meta-objeto associado permite a enumeração e busca das interfaces de um objeto, bem como dos métodos e atributos de uma determinada interface.
- Modelo de Arquitetura: permite a reificação de um determinado componente em termos de seus componentes internos e da forma como os mesmos estão interconectados. O protocolo de meta-objeto permite a inspeção da composição de um determinado objeto, bem como alterações nesta composição, através da inserção, remoção ou substituição de objetos componentes. Desta forma, é possível realizar mudanças na implementação interna de um objeto em tempo de execução.
- Modelo de Interceptadores: permite alterar o comportamento de uma determinada interface através da inserção de interceptadores em seu caminho de invocação. Interceptadores podem ser utilizados, por exemplo, para introduzir propriedades não funcionais à interface, tais como segurança e controle de qualidade de serviço. O respectivo protocolo de meta-objeto permite enumerar os interceptadores associado a uma interface, bem como adicionar ou remover interceptadores.
- Modelo de Recursos: permite reificar o conjunto de recursos utilizados pelos objetos da plataforma, tais como armazenamento e processamento. O protocolo de meta-objeto associado permite a inspeção dos recursos utilizados, bem como a alteração dinâmica das propriedades e quantidades para cada tipo de recurso.

As soluções de *middleware* DynamicTAO e o OpenORB tiveram motivações e resultados similares. As principais características dessas duas soluções estão sumarizadas no Quadro 1.

Quadro 1 - Comparação entre o DynamicTAO e o OpenORB

DynamicTAO	**OpenORB**
Começou com o TAO	Começou do zero.
TAO foi modular, porém ainda estático	Uma nova arquitetura de *middleware*
Adiciona princípios de reflexão	Adiciona princípios de reflexão
Mais flexível	Menos flexível
Mais dinâmico	Menos dinâmico
Mais customizável	Menos customizável

C. CARISMA [Capra, L. et. al 2003]

O projeto CARISMA é um modelo de *middleware* que explora a reflexão de modo a habilitar interações sensíveis ao contexto entre aplicações móveis [9].

O modelo reflexivo apresentado na Figura 9 é utilizado na construção de *middlewares* reflexivos para aplicações móveis. Note que temos dois processos distintos: a reificação e a absorção.

A reificação consiste na disponibilização de aspectos internos do *middleware* através de meta informações, de modo que as aplicações possam, dinamicamente, verificar o comportamento do *middleware* e alterá-lo através de uma meta interface, o que caracteriza o processo de absorção.

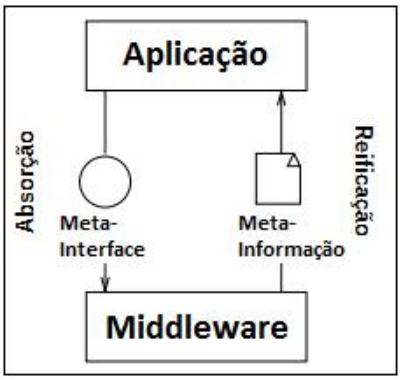

Figura 9 - Modelo do processo reflexivo – CARISMA

O artigo também trata de conflitos, que podem ocorrer quando diferentes políticas são acionadas no mesmo contexto para entregarem determinado serviço, fazendo com que o *middleware* não saiba qual aplicar. São apresentados e discutidos os tipos de conflitos e citados exemplos de como evitá-los ou, em caso necessário, contorná-los. Baseados em técnicas microeconômicas, os autores propõem um mecanismo para resolução de conflitos e percorrem todos os passos para aplicá-lo, desde a definição do protocolo até o momento em que são apresentados exemplos e a forma como eles são avaliados pelo mecanismo proposto.

D. MobiPADS [Chan, A. et. al 2003]

O MobiPADS (*Mobile Plataform For Actively Deployable Service*) [10] é um *middleware* reflexivo ciente de contexto e que provê uma plataforma de execução para habilitar a implantação de serviços ativos e a reconfiguração da composição de serviço em diversos ambientes. MobiPADS suporta a adaptação dinâmica tanto do *middleware* quanto das aplicações para fornecer uma configuração flexível dos recursos para otimização das aplicações móveis em um ambiente sem fio.

A plataforma do MobiPADS é composta de dois agentes: um servidor com fio ligado à rede e um cliente atrelado ao dispositivo móvel através de uma rede sem fio ou de uma rede de celular. Os dois agentes são responsáveis por verificar o tráfego sobre o *link* sem fio e fornecer um ambiente de operação ideal para aplicações móveis.

Dentre os componentes do *middleware* podem-se citar: um gerenciador de configuração, um gerenciador de migração de serviço, um diretório de serviço, um registrador de eventos e um servidor de canais. A Figura 10 traz uma visão geral da arquitetura do sistema MobiPADS.

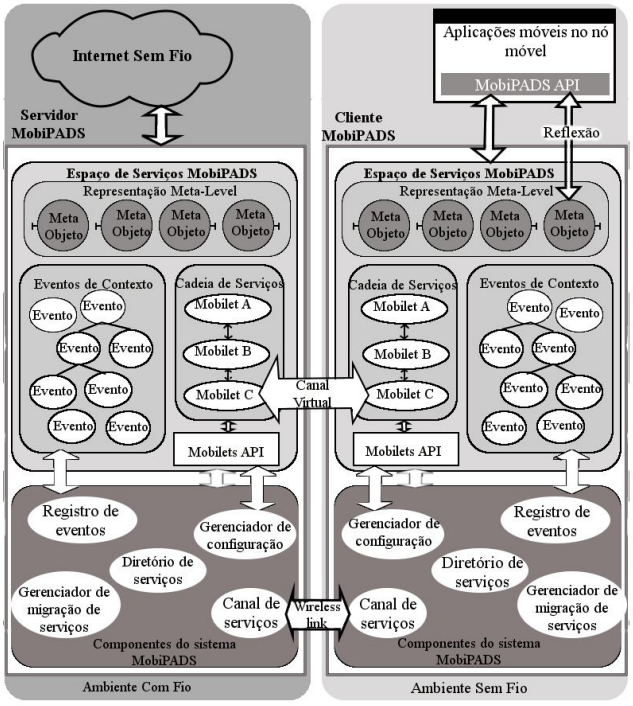

Figura 10 - A arquitetura do sistema MobiPADS

E. ReMMoc [Grace, P. et. al 2003]

Para operar em ambientes dinâmicos e potencialmente desconhecidos, um cliente móvel deve primeiro descobrir os serviços locais que correspondem a suas necessidades, e então interagir com estes serviços para obter a funcionalidade dos aplicativos. Essa alta dinamicidade e a natureza desconhecida do ambiente pode ser um problema nos processos de descoberta dinâmica,

O *Framework* ReMMoc (*Reflective Middleware for Mobile Computing*) [11] é um *middleware* reflexivo adaptado ao problema de descoberta em ambientes dinâmicos e potencialmente desconhecidos que tem como operação chave a descoberta dinâmica e protocolos de interação para mapear o ambiente de serviço móvel corrente.

A arquitetura do ReMMoc se reconfigura automaticamente para a combinação com o serviço atual e é projetado para operar sobre dispositivos móveis, tais como *laptops* e telefones celulares.

Uma das camadas do *framework* ReMMoc é de alto nível, na qual um conjunto de componentes são "plugados" na mesma. Os autores trazem três direções para essa camada de alto nível: A primeira direção é uma seção concreta, no qual é composta por um *framework* de ligação para interoperabilidade com os serviços móveis de diferentes tipos e um serviço de descoberta para descoberta de serviços em um determinado raio de ação.

A segunda direção é um modelo abstrato de programação que implementa uma API para realização do serviço de descoberta. Já a terceira e última direção é uma seção para mapeamento do abstrato para o concreto, no qual consiste em componentes que mapeiam as requisições de serviço. Note na Figura 11 a arquitetura geral do *framework* ReMMoc.

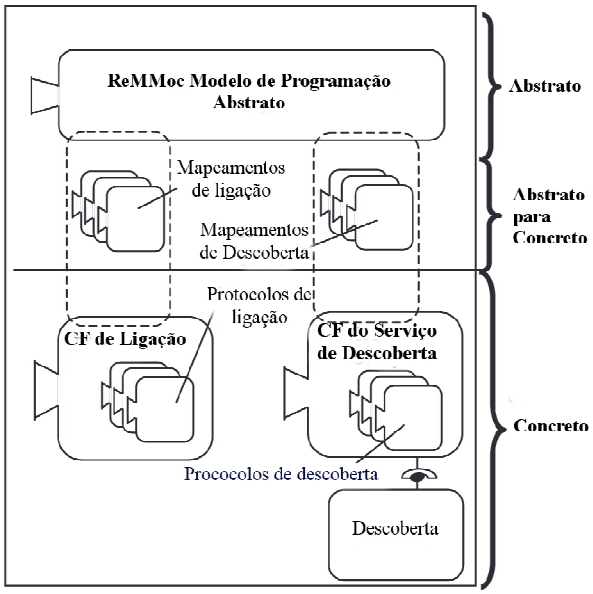

Figura 11 - A arquitetura do *framework* ReMMoc

F. CARM [Liu, S. et. al 2011]

Uma das técnicas para construção de sistemas embarcados e distribuídos de tempo real é a utilização de *middlewares* reflexivos cientes de contexto (*Context-Aware Reflective Middleware* - CARM). Esses *middlewares* suportam a reconfiguração das aplicações e a adaptações às mudanças de comportamento em tempo de execução [12].

O *framework* CARM pode monitorar informações de contexto em tempo real e adaptar as aplicações a essas mudanças de contexto. Ele provê uma abordagem de reconfiguração para construção de sistemas embarcados e distribuídos de tempo real (*Distributed Real-Time and Embedded Systems* - DRE).

Segundo [12], o *framework* CARM possui restrições de dependência e propicia um atraso significativo na reconfiguração das aplicações, o que o torna inviável na satisfação dos rigorosos requisitos das aplicações DRE.

Nesse sentido os autores propõem um *middleware* para colaboração robusta adaptativa entre sistemas e ambientes heterogêneos (*Middleware for Adaptive Robust Collaborations across Heterogeneous Environments and Systems* - MARCHES), no qual oferece uma estrutura de múltiplas cadeias de componentes para redução do tempo das mudanças de comportamento locais e um protocolo que utiliza mensagens ativas para redução do tempo de sincronização do comportamento distribuído.

A reflexão é fornecida tanto em nível de componentes quanto em nível do sistema. Para avaliação do contexto, MARCHES utiliza um modelo de notificação de eventos hierárquico. Já a descrição e o gerenciamento das políticas de adaptação são realizados por meio de uma linguagem baseada em XML (*Extensible Markup Language*).

G. MINA [Qin, Z. et. al 2014]

Segundo [13], devido a grande diversidade das tecnologias de acesso, tais como *WiFi*, *Ethernet*, *ZigBee*, entre outras, o controle da infraestrutura de

todas essas possibilidades de rede torna-se uma tarefa desafiadora e bastante complexa. Nesse contexto, os autores propõem uma Arquitetura de Informação de Multi-rede (*Multinetworking INformation Architecture* – MINA), uma abordagem de *middleware* reflexivo para o gerenciamento dinâmico de diferentes redes heterogêneas em ambientes pervasivos.

Um aspecto no uso do *framework* MINA é que o mesmo implementa a filosofia de projeto observar-analisar-adaptar (*Observe-Analyze-Adapt* – OAA) para guiar a configuração, o gerenciamento do estado, e a coordenação da multi-rede, utilizando o conhecimento prévio do estado da rede. Note na Figura 12 o funcionamento do paradigma OAA no *framework* MINA.

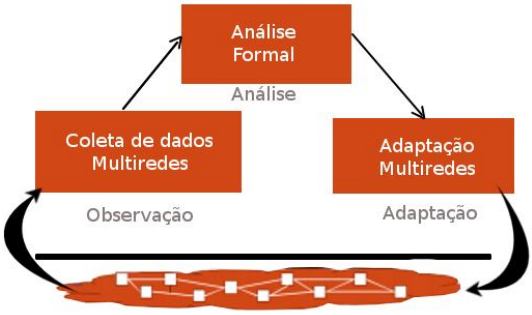

Figura 12 - O Paradigma OAA

O principal objetivo do passo "Observar" é gerar uma árvore baseada no atraso da rede para coletar informações de estado dos vários componentes existentes no ambiente de multi-rede. Para avaliar o desempenho da rede, os autores propõem o uso do método formal de *Maude*, uma linguagem de especificação executável para explorar o gerenciamento proativo das informações coletadas durante o passo de observação. No passo "Adaptação" as mudanças são realizadas, por o ambiente de multi-rede ser bastante dinâmico, as adaptações ocorrem com mais frequência que em ambientes de redes tradicionais.

IV. Um Estudo Compartivo entre as Soluções de Middlewares Reflexivos

Na seção anterior vimos algumas das principais soluções de *middlewares* reflexivos utilizados para permitir a inspeção e a adaptação das aplicações com alto grau de dinamicidade. Esta seção é responsável por reunir as principais características dessas soluções de *middleware* em um quadro comparativo.

Para efeito de comparação consideramos as seguintes características no quadro comparativo: Palavras-Chave, Arquitetura Básica, Técnicas de Reflexão, Desenvolvimento, Orientado a Componentes, Qualidade de Recursos e Ambiente de Aplicação.

A coluna "Palavras-Chave" indica quais são os principais termos utilizados no artigo. A coluna "Plataforma" mostra qual é a tecnologia básica de *middleware* utilizada. A coluna "Desenvolvimento" indica se a solução ainda é desenvolvida ou se o estágio de desenvolvimento já chegou ao fim. A coluna denominada "Orientado a Componentes" indica o nível de orientação a componentes do *middleware*. A coluna "Qualidade de Recursos" mostra o nível da qualidade de recursos oferecido pelo *middleware*. Por fim, coluna "Ambiente de Aplicação" refere-se a qual tipo de rede e para quais tipos de aplicações a solução em questão é adequada. O Quadro 2 mostra a relação entre as diferentes soluções de *middlewares* reflexivos abordados neste trabalho.

O estudo comparativo trás informações importantes sobre características comuns entre os principais trabalhos sobre *middlewares* reflexivos, além de permitir que novas tomadas de decisão sejam tomadas sobre o uso desta categoria de *middleware* em uma variedade de ambientes, como naqueles encontrados em IoT.

As técnicas de inspeção e adaptação são usadas de forma unânime em todos os trabalhos analisados. Nota-se também que o gerenciamento dinâmico é necessário para aplicações com alto grau de dinamicidade. O uso de diferentes técnicas tais como a reificação e a absorção, pode ser utilizado para aplicações que demandam requisitos de tempo real.

V. Desafios de Pesquisa

Existem diversos problemas que precisam ser resolvidos para correta construção e utilização de arquiteturas de *middlewares* baseados em componentes. Alguns dos desafios a serem superados são listados:

- Todas as terminologias relevantes devem ser claramente e explicitamente definidas, tais como: componentes, serviços e aplicações.
- É necessário conhecer claramente o comportamento de todos os componentes e de suas dependências. Esse conhecimento é muito importante para manutenção da integridade do sistema para a correta atualização dos componentes.
- Algumas funções de monitoramento são requeridas, e a arquitetura projetada deve suportar gerência local e remota. Se a gerência remota for alcançada, pode-se facilmente integrar novas funções para os componentes através de *links* remotos.

Diversos são os desafios existentes quando o assunto são *middlewares* e sistemas distribuídos em geral: a heterogeneidade dos diversos dispositivos, devido ao crescimento do interesse da indústria em criar sistemas ubíquos e a necessidade de fornecer sistemas cada vez mais escaláveis, é um dos principais

desafios enfrentados pelos programadores de *middlewares* [13].

Alguns problemas podem ocasionar queda de desempenho das aplicações para a Internet, sendo necessário aos desenvolvedores de *middlewares* reavaliarem algumas arquiteturas, tais como CORBA e DCOM, para a perspectiva de IoT. Essas soluções devem lidar com questões mais complexas, tais como a autonomia, a autoridade descentralizada, a conectividade intermitente e a evolução contínua da escalabilidade.

Os avanços na tecnologia acontecem com frequência. Devido a essa característica, as aplicações desenvolvidas se tornam rapidamente obsoletas e precisam ser melhoradas ou até mesmo substituídas. Por mais que os *softwares* evoluam, chega um momento em que o *hardware* começa a atrapalhar os avanços, ocasionando, como já citado anteriormente, problemas de conectividade e principalmente de escalabilidade.

A simples representação dos requisitos de tempo real de forma adequada para que seja realizada a introspecção e adaptação das aplicações é um dos primeiros desafios na construção de *middlewares* reflexivos. Os requisitos de reflexão dependem da representação dos requisitos do sistema de tempo real.

Um objetivo importante nos requisitos de reflexão é a permissão do sistema auto-adaptativo raciocinar sobre ele mesmo e reavaliar as suas necessidades durante a execução. Quaisquer reavaliações sobre os requisitos devem, naturalmente, ser refletidas no sistema em tempo de execução. Um aspecto importante para que isso ocorra é a sincronização entre a representação dos requisitos de tempo real e a arquitetura de *software*.

Uma forte característica de *middlewares* reflexivos é a habilidade de perceber o que está acontecendo em sua volta, colher e selecionar as informações mais concisas e ainda se adaptar para atender o que deles se espera. A descoberta e seleção de fontes de contexto são apontadas como um dos desafios que devem ser tratados quando o assunto é *middleware* reflexivo sensível ao contexto.

Por fim, o próprio desafio de lidar com o incerto devido à natureza estocástica dos eventos do ambiente, das capacidades limitadas de alguns dispositivos, e das dificuldades na predição do resultado da modificação dos serviços no comportamento das aplicações e nos objetivos do sistema faz com que a construção de *middlewares* seja uma tarefa bastante complexa e desafiadora [14].

VI. Conclusões e Trabalhos Futuros

As plataformas de *middlewares* convencionais não suportam alguns requisitos inerentes a aplicações móveis e multimídias, caracterizadas por um alto grau de dinamicidade. Sabe-se que o ambiente de execução pode variar de maneira imprevisível. Dessa forma, as plataformas de *middleware* devem ter suporte a essa característica. Os *middlewares* reflexivos conseguem acompanhar essa dinamicidade através das adaptações realizadas em sua própria estrutura [15].

São vários os *middlewares* reflexivos existentes e, com a crescente popularização do conceito de computação ubíqua, a tendência é que surjam cada vez mais aplicações com essas características: mobilidade e dinamicidade. Neste trabalho conceituamos e detalhamos *middlewares* reflexivos com base na visão de alguns dos principais trabalhos da área. Para cada *middleware* apresentado, destacamos as principais características e o funcionamento geral da solução proposta pelos autores.

Como resultado, apresentamos um estudo comparativo, resumido no Quadro 2, no qual é notável que CORBA e Java são os principais alicerces de soluções de *middlewares* reflexivos existentes. As técnicas de inspeção e adaptação trabalham em conjunto de modo a fornecer princípios de reflexão e alguns deles são voltados para aplicações que possuem mobilidade.

Em relação aos trabalhos futuros pretendemos detalhar uma proposta de um *middleware* reflexivo sensível ao contexto que será utilizado como interface para aplicações ubíquas e cientes de contexto aplicadas ao paradigma de IoT.

Referências

[1] Kon, F., Costa, F., Blair, G. e Campbell, R. H. (2002) “The Case for Reflective Middleware”, In: Communications of the ACM, Vol. 45, No. 6.

[2] Weiser, M. (1999). “The computer for the 21st century”. In: SigMobile Mob.Co.

[3] Agha, G. A. (2002) "Adaptive Middleware", In: Communications of the ACM, Vol. 45, No. 6, pp. 31-32.

[4] Maes, P. (1987). “Concepts and experiments in computational reflection”. In: ACM Sigplan Notices.

[5] Provensi, L. L. (2009) ”Uma plataforma de middleware reflexivo com suporte para auto-adaptação”. Dissertação de Mestrado. Universidade Federal de Goiás.

[6] Kon, F., Roman, M., Liu, P., Mao, J., Yamane, T., Magalhaes, L., and Campbell, R. (2000) “Monitoring, security, and dynamic configuration with the dynamicTAO reflective ORB”. In: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing.

[7] Blair, G., Coulson, G., Andersen, A., Blair, L., Clarke, M., Costa, F., Duran-Limon, H., Fitzpatrick, T., Johnston, L., Moreira, R., Parlavantzas, N., e Saikoski, K. (2001) “The design and implementation of OpenORB 2”. In: IEEE Distributed Systems.

[8] Costa, F. (2004). "Plataforma de Middlware Reflexivo para Suporte a Aplicações Dinâmicas em Ambientes Móveis. Projeto de Pesquisa do Instituto de Informática. Universidade Federal de Goiás.

[9] Capra, L., Emmerich, W., e Mascolo, C. (2003). “CARISMA: Context-aware reflective middleware system for mobile applications” In: Software Engineering, IEEE Transactions on, 29(10), 929-945.

[10] Chan, A. T., e Chuang, S. N. (2003). “MobiPADS: a reflective middleware for context-aware mobile computing”. In: Software Engineering, IEEE Transactions on, 29(12), 1072-1085.

[11] Grace, P., Blair, G. S., & Samuel, S. (2003). ReMMoC: A reflective middleware to support mobile client interoperability. In On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE (pp. 1170-1187).

[12] Liu, S., e Liang, C. (2011) "A context-aware reflective middleware framework for distributed real-time and embedded systems." In: Journal of Systems and Software, pages 205-218.

[13] Qin, Z., Iannario, L., Giannelli, C., Bellavista, P., Denker, G., e Venkatasubramanian, N. (2014). “MINA: A Reflective Middleware for Managing Dynamic Multinetwork Environments”. In: Proceedings of IEEE/IFIP Network Operations and Management Symposium.

[14] Bencomo, N., Whittle, J., Sawyer, P., Finkelstein, A., e Letier, E. (2010). “Requirements reflection: requirements as runtime entities”. In: Proceedings of the International Conference on Software Engineering.

[15] Kurt, G. (2001) “Middleware Challenges Ahead”. In: IEEE Computer Society.

Quadro 2 - Um estudo comparativo entre as principais soluções de *Middlewares* Reflexivos.

Trabalhos Selecionados	**Palavras-Chave**	**Principais Conceitos**	**Plataforma**	**Técnicas de Reflexão**	***Status* do Desenvolvimento**	**Orientado a Componentes**	**Qualidade de Recursos**	**Ambiente de Aplicação**
OpenORB [7]	Meta espaço	Arquitetura e Interface Reflexiva	CORBA	Inspeção e Adaptação	Terminado	Sim	Médio	Aplicações com Requisitos Dinâmicos
DynamicTAO [6]	Reificação	Descrição Interna Explícita	CORBA	Inspeção e Adaptação	Terminado	Sim	Alto	Aplicações com Requisitos de Tempo Real
CARISMA [9]	Reificação e Absorção	Conflitos, Meta-Informação e Meta-Interface	___	Inspeção e Adaptação	Terminado	Não	Alto	Aplicações Dinâmicas e Móveis
MobiPADS [10]	Agentes e Ambiente de Operação	Serviços Ativos e Recomposição de Serviços	JAVA	Inspeção e Adaptação	Terminado	Sim	Alto	Aplicações Dinâmicas e Móveis
ReMMoc [11]	Reflexão	Descoberta dinâmica e mapeamento de serviço	___	Inspeção e Adaptação	Terminado	SIm	Alto	Aplicações Dinâmicas e Móveis
CARM [12]	Reconfiguração e Sincronização	Reconfiguração Eficiente em Aplicações de Requisitos de Tempo Real	___	Inspeção e Adaptação	Terminado	Sim	Alto	Aplicações Dinâmicas e Móveis
MINA [13]	Gerenciamento Dinâmico	OAA (Observar-Analisar-Adaptar)	JAVA	Inspeção e Adaptação	Terminado	Não	Alto	Diversas Aplicações em Redes Heterogêneas
Proposta	Inspeção e Adaptação Dinâmica	Reificação e Adaptação	JAVA	Inspeção e Adaptação	Em andamento	Sim	Alto	Cenário de IoT