CienteFico. Ano IV, v. I, Salvador, janeiro-junho 2004

Middleware
Uma solução para o desenvolvimento de aplicações distribuídas

Rita Suzana Pitangueira Maciel, Semírames Ribeiro de Assis[1]

Resumo

O termo *middleware* representa uma camada intermediária entre o sistema operacional e as aplicações distribuídas, tendo como objetivo abstrair a heterogeneidade existente da comunicação distribuída. Existem alguns tipos de *middleware*, assim como algumas categorias, que serão discutidos neste artigo. Através de uma pesquisa bibliográfica, busca-se conceituar o termo *middleware*, assim como suas vantagens e desvantagens. Conclui-se que *middleware* tem um grande potencial para facilitar a construção de aplicações distribuídas.

Palavras – chave:

Middleware, Aplicações Distribuídas, Integração entre Sistemas Heterogêneos

O termo *middleware* caracteriza uma camada de *software* que possibilita comunicação entre aplicações distribuídas - tendo por objetivo diminuir a complexidade e heterogeneidade dos diversos sistemas existentes -, provendo serviços que realizam a comunicação entre esta categoria de aplicações de forma transparente às mesmas. A

1 Rita Suzana Pitangueira **Maciel** (**rsuzana@frb.br**) é professora do Curso de Ciência da Computação na Faculdade Ruy Barbosa e Semíramis Ribeiro de Assis (semiramis@frb.br) é estudante do Curso de Ciência da Computação da Faculdade Ruy Barbosa

adaptação entre sistemas heterogêneos é necessária, por exemplo, quando um sistema atual deve interoperar com sistemas obsoletos ou com diferentes empresas. Um *middleware* é dividido em componentes do ambiente de programação e do ambiente de execução [15].

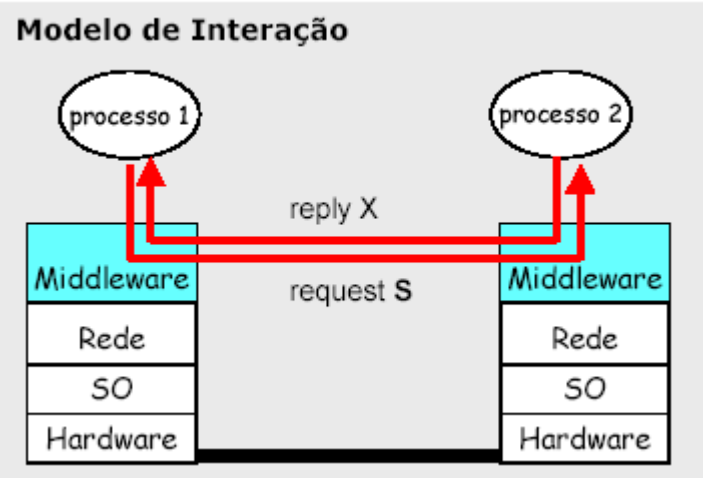

Figura 1 – Comunicação através de *middleware* [15].

A figura 1 ilustra o processo de comunicação entre aplicações distribuídas através de um *middleware*. Todo o processo é transparente para as aplicações, e a heterogeneidade existente é tratada também pelo *middleware*.

Inicialmente, um *middleware* tinha a função básica de unir os componentes de um programa distribuído, ditando a maneira pela qual estes componentes interoperavam. Atualmente, sua função é integrar aplicações completas entre e dentro de organizações [3].

No que se refere à integração entre aplicações escritas em diferentes linguagens de programação, poucos *middlewares* suportam esta característica. Um bom exemplo de integração entre diferentes linguagens é o *Commom Object Request Broker Architecture* (CORBA), pois neste *middleware* é possível fazer o mapeamento de uma *Interface Definition Language* (IDL) em muitas linguagens de programação [2].

Um ponto importante na utilização e funcionalidade dos *middlewares* é a sua padronização, o que causa problemas relacionados à integração, facilidade de uso e gerenciamento. A

padronização, entretanto, é importante para auxiliar os consumidores a selecionarem *middlewares* baseados em qualidade.

Duas características importantes na construção de um *middleware* são sua flexibilidade e performance. Porém, estes dois pontos são mutuamente exclusivos, já que o alto nível de flexibilidade acaba impedindo um alto nível de performance, sendo o ideal um balanceamento entre os dois.

Outro ponto a ser destacado na utilização dos *middlewares* é a redução na complexidade do sistema, pois, ao utilizar um *middleware*, o sistema terá sua complexidade reduzida, sendo que este é um dos objetivos principais na utilização desta camada intermediária.

Atualmente, o principal desafio enfrentado pelo *middleware* é a facilitação da integração entre aplicações para Internet. Assim como a integração entre consumidores e a Internet está acontecendo a cada dia que passa, é importante que haja integração entre aplicações via este mesmo meio de acesso, vencendo, assim, as limitações dos *browsers* no que se refere ao modelo de aplicações de duas camadas.
O artigo está dividido da seguinte forma: a próxima seção abordará os serviços oferecidos pelos *middlewares*. A seguir, tem-se a classificação dos *middlewares* no que se refere a tipo e categoria. Por fim, tem-se a conclusão e a referência bibliográfica.

SERVIÇOS DO *MIDDLEWARE*

Na definição de Bernstein [4], o serviço oferecido pelo *middleware* é um serviço de propósito geral, situado entre plataformas (serviços de baixo nível) e aplicações, sendo caracterizado pelas APIs (*Application Programmer Interface*) e pelos protocolos que suporta.

Devido às propriedades dos componentes de um *middleware*, seus serviços não são dependentes de plataforma ou aplicação, sendo genéricos entre as diversas aplicações dos diversos fabricantes, suportando interfaces e protocolos padrões [4].

Para que seja considerado um *middleware*, o serviço oferecido deve atender às necessidades de uma gama de aplicações de um determinado domínio, não se concentrando apenas em serviços fornecidos para uma aplicação específica, além de ter implementação independente de plataforma. Esta independência de plataforma é conseguida no momento da implementação dos serviços, quando se busca atingir a portabilidade, o que significa que o *middleware* poderá ser transferido para outra plataforma com o mínimo esforço [4].

Um serviço fornecido por um *middleware* deve dar suporte a pelo menos uma API padrão, sendo considerado transparente em relação a esta API se puder ser acessado por ela sem a modificá-la. Porém, o que se encontra são serviços implementados sobre API's proprietárias ou protocolos que ainda não foram oficialmente publicados, fato que dificulta a padronização entre os diferentes vendedores. Os *middlewares* comumente fornecem os seguintes serviços:

- **Gerenciamento de Apresentação:** Gerenciamento de formulários, gráficos, impressora e hipermídia.
- **Computação:** Ordenação, dispositivos matemáticos, serviços internacionalizáveis, conversores de dados e serviço de tempo.
- **Gerenciamento de informação:** Servidor de diretórios, gerenciador de *log*, gerenciador de arquivos.
- **Comunicação:** Mensagens *Peer-to-Peer*, chamada remota de procedimento, fila de mensagens, mensagem eletrônica e exportação eletrônica de dados.
- **Controle:** Gerenciamento de transações, de *threads*.
- **Gerenciamento do Sistema:** Serviço de notificação de eventos, serviço de contas, gerenciamento de configuração, detector de falhas.
- **Sistema de entrega:** Um padrão proposto é o *Java Virtual Machine* (JVM) para o sistema de entrega. Outro exemplo é o *Remote Procedure Call* (RPC).

- **Comunicação entre processos:** É o coração do *middleware*. Como exemplo pode-se citar o *Object Request Broker* (ORB) do CORBA, que tem por objetivo a integração entre aplicações remotas.
- **Interface com o usuário:** Como exemplo tem-se o HTML, e os formatos multimídia aceitos pela Internet [5].
- **Em direção de uma utilização mais universal de *middlewares* comuns:** Os esforços para a criação de *middleware* estão centrados, atualmente, em soluções proprietárias e direcionadas a algum interesse. Os resultados disso ainda são funcionalidades caras e proibitivas. Uma das razões pelas quais o reuso ainda é utilizado é a facilidade de agrupar soluções direcionadas, que permanece invisível a todos menos ao desenvolvedor. Uma solução para este problema é a conscientização dos programadores em relação ao custo do ciclo de vida de desenvolvimento que está por trás da interconexão de componentes individuais [14].
- **Soluções comuns suportando tanto variabilidade quanto controle:** Os sistemas tendem a trabalhar de uma maneira adequada se tiverem todos os recursos necessários para o seu correto funcionamento. Há pouca ou nenhuma flexibilidade em seus comportamentos. O que é necessário, então, é uma reconfiguração do sistema em relação aos recursos disponíveis a eles, que tem dois focos: o comportamento individual e o agregado [14].

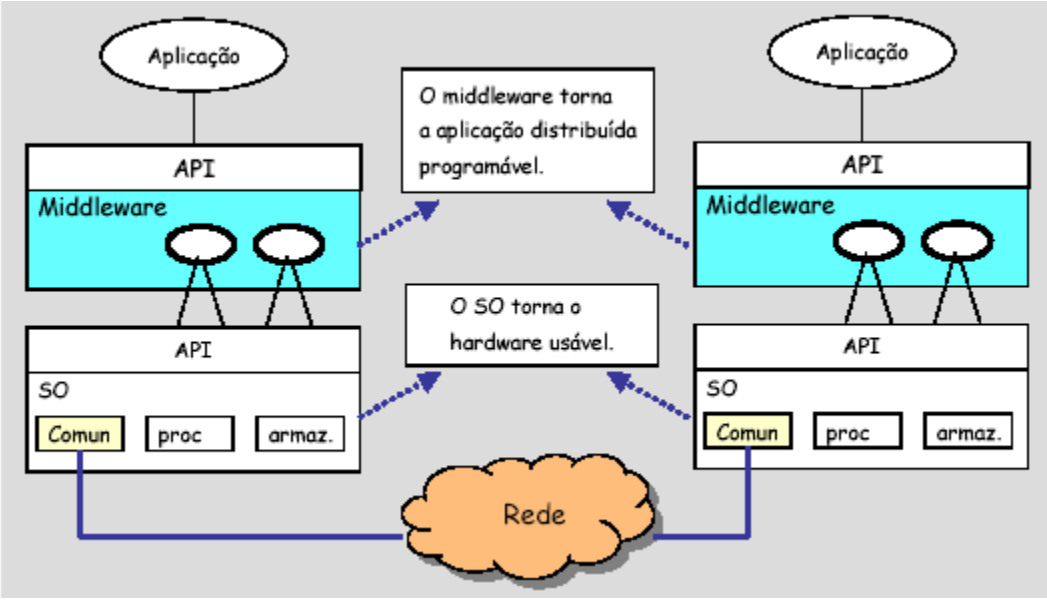

Figura 2 – Contexto do *middleware* [15].

A figura 2 ilustra o contexto do *middleware* na comunicação entre aplicações distribuídas e o sistema operacional. Um *middleware* utiliza API's fornecidas pelo sistema operacional (S.O) para que a comunicação distribuída seja facilitada. A responsabilidade do S.O é apenas gerenciar o funcionamento do *hardware*.

Outros serviços incluídos são gerenciamento de qualidade de serviço (QoS) e segurança da informação. Apesar da integração entre QoS e *middleware* ser importante, é necessário que haja uma padronização para que este procedimento possa ser realizado [7].

CLASSIFICAÇÃO DOS *MIDDLEWARES*

A utilização de *middlewares* não é transparente ao desenvolvedor. Alguns fatores indicam esta falta de transparência:

- a comunicação entre objetos distribuídos é lenta se comparada à comunicação entre objetos locais;
- a ativação e desativação de componentes gera a necessidade da implementação de persistência destes componentes;
- os componentes devem ser desenvolvidos de maneira que possam lidar com as interações concorrentes que ocorrem em componentes distribuídos;
- os componentes podem escolher entre as primitivas de sincronização que são oferecidas pelo *middleware* e a necessidade da exploração correta destas.

    - Segundo Talarian [6], os *middlewares* são classificados nas seguintes categorias:

- **Monitor de Processamento de Transação:** provê um ambiente completo para aplicações de transação que acessam banco de dados relacionais. O *overhead* de comunicação neste modelo fica reduzido ao mínimo devido à troca de mensagens se basear em simples *request/reply*.

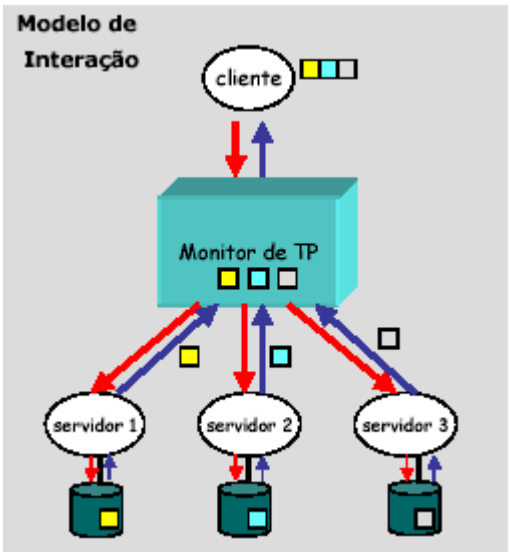

Figura 3. Modelo de interação do *middleware* orientado a transação [15].

- ***Remote Procedure Call* (RPC):** Uma das primeiras formas de comunicação entre processos remotos, operando em baixo nível. Não se aplicam bem a aplicações grandes e em tempo real.

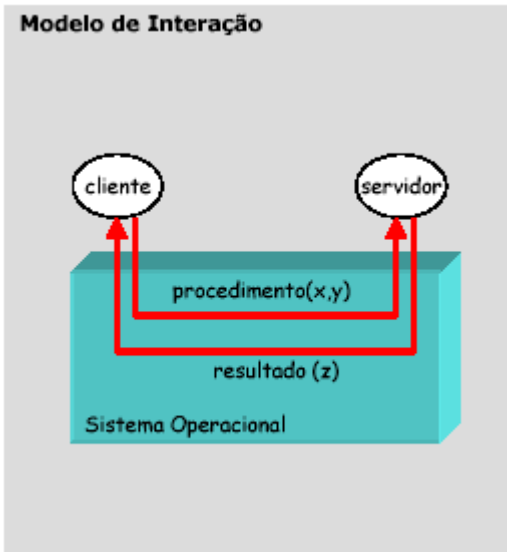

Figura 4. Modelo de interação do *middleware* RPC [15].

- ***Object Request Broker* (ORB):** Pode ser considerado um RPC orientado a objetos. Existem dois concorrentes: CORBA, da Object Management Group (OMG), e DCOM, da Microsoft.

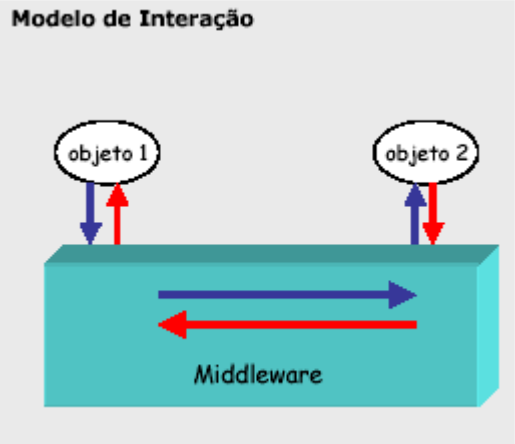

Figura 5. Modelo de interação do *middleware* orientado a objetos [15].

- *Homegrown Middleware:* Este tipo de *middleware* se destina a aplicações específicas, ou seja, que são feitas para resolver um problema específico. Sua constante atualização e personalização o tornam caro e com flexibilidade e flexibilidade afetadas.
- Orientado à mensagem (MOM): Funciona com base na troca de mensagens entre programas de maneira assíncrona.

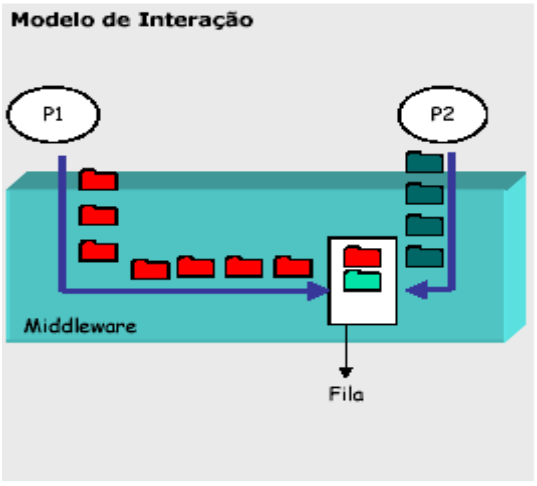

Figura 6. Modelo de interação do *middleware* orientado a mensagem [15].

- **Serviço de mensagem Java (JMS):** Especifica um conjunto de interfaces pelas quais programas em Java podem acessar softwares orientados a mensagem.

Tipos de *middleware*

Alguns tipos de *middleware* que serão descritos abaixo são: reflexivo, adaptativo e *Distributed Object Computing* (DOC).

Middleware Reflexivo

Um *reflective middleware* utiliza o conceito de refletividade computacional, ou seja, uma aplicação pode acessar algumas partes do estado do sistema logo abaixo e modificá-lo dinamicamente, modificando, então, sua própria semântica [9].

O uso da refletividade de maneira indisciplinada pode resultar em quebras inesperadas do sistema, principalmente se a modificação resulta em mudanças incompatíveis para uma determinada parte da aplicação [9].

Segundo Román [10], um *middleware* reflexivo explora o protocolo meta-objeto de Gregory Kiczales, combinado às idéias de refletividade computacional e orientação a objetos, sendo dividido em *base-level* e *metalevel*..

A idéia base do *base-level* é atingir a funcionalidade das aplicações, enquanto que o *metalevel* designa colações de componentes que constituem a arquitetura interna da plataforma do *middleware.* A propriedade refletiva permite que o comportamento destes objetos seja monitorado, possibilitando mudanças no comportamento do *middleware* [10].

A aplicabilidade de um *middleware* reflexivo se concentra em aplicações de computação onipresente em 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 da computação onipresente, característica essa não existente em plataformas de *middleware* convencionais, por serem grandes e inflexíveis [10].

Segundo Korn [11], o *middleware* reflexivo é implementado como um conjunto de componentes colaborativos que podem ser configurados ou reconfigurados pela aplicação, tendo sua interface imutável e podendo suportar aplicações desenvolvidas para *middlewares* tradicionais.

A importância dos metadados e do uso da refletividade na estrutura do *middleware* é que estes armazenam informações sobre o estado das aplicações que estão rodando sobre o *middleware*, de maneira que cada aplicação específica possui um perfil próprio, o qual, ao ser modificado, atingirá a própria aplicação e o comportamento do *middleware*. De maneira mais direta, a função dos metadados é ditar ao *middleware* o seu comportamento quando uma determinada ação for executada, enquanto que a refletividade é utilizada para armazenar perfis específicos para cada aplicação [12].

Middleware Adaptativo

O objetivo desta categoria de *middleware* é ser dinamicamente personalizável, o que possibilita às aplicações móveis a flexibilidade necessária [9].

Segundo Yau [13], apesar da natureza da adaptação de aplicações ser específica por aplicação, cabe ao *middleware* prover um *framework* flexível para o desenvolvimento de aplicações adaptativas e sensíveis ao contexto. O *middleware* adaptativo deve ser reconfigurável, permitindo que novos serviços sejam acoplados. Esta categoria de *middleware* deve ter as seguintes características:

- **Suporte a aplicações sensíveis ao contexto e adaptativas:** Facilita na construção de aplicações adaptativas e sensíveis ao contexto por um *middleware*.
- **Suporte em tempo de execução de comunicações *ad hoc* entre as aplicações:** Considerando que o custo da comunicação em ambientes sem fio é maior que o

custo da computação neste mesmo ambiente, cabe ao *middleware* utilizar de maneira inteligente o recurso da comunicação baseado no contexto e nas necessidades reais das aplicações.

- **Serviços adaptativos específicos da aplicação:** Além dos serviços disponibilizados pelo *middleware*, uma aplicação pode ter serviços específicos, que envolvem segurança e gerenciamento de grupo.
- **Interoperabilidade com *middleware* de outros domínios:** Com o objetivo de fornecer a interoperabilidade necessária com *middlewares* de outros domínios, o *middleware* adaptativo deve ter suporte à interoperabilidade, o que permite que aplicações móveis se comuniquem com as aplicações já existentes sem muito esforço.

Distributed Object Computing Middleware (DOC)

Este tipo de *middleware* se destina a aplicações distribuídas, e, como em um protocolo de rede que se divide em múltiplas camadas, também possui uma arquitetura decomposta em camadas que são, de acordo com Schantz [14]:

- ***Host Infrastructure middleware:*** Encapsula a comunicação pertencente ao SO e o mecanismo de concorrência, criando, então, componentes reusáveis da programação distribuída. Estes componentes, além de ajudar na encapsulação de individualidades do sistema operacional, também auxilia na obtenção de aplicações menos susceptíveis a erro e com uma maior portabilidade entre plataformas diferentes. Alguns exemplos de *middlewares* pertencentes à esta camada são: a máquina virtual Java da Sun (JVM), a plataforma .NET da Microsoft para Web Services.
- ***Distribution Middleware:*** Define modelos de programação de auto-nível dos quais as API's e os componentes podem automatizar e estender a capacidade de programação já existente no sistema operacional. *Middlewares* pertencentes à esta camada proporcionam a construção de aplicações distribuídas como se fossem aplicações locais. Dentre estes *middlewares* se encontram: CORBA [16], RMI [17], DCOM [19] e SOAP [18].

- ***Commom Middleware Services:*** Define serviços de alto nível que podem ser utilizados pelos programadores para a construção de aplicações distribuídas. Estes serviços permitem que o programador não se preocupe com a forma com que sua aplicação irá se comunicar com sua parte distribuída, focando mais a lógica desta aplicação. Exemplos destes serviços são: os serviços CORBA, *Sun's Enterprise Java Beans* (EJB), .NET Web Services.
- ***Domain-specific middleware services***: São serviços específicos requeridos por determinados domínios. Ao contrário das outras camadas do DOC, que provêem mecanismos de reuso e serviços horizontais, este tipo de serviço possui seu alvo no mercado vertical. Os serviços oferecidos por esta camada possibilitam um maior crescimento da qualidade do sistema e diminuem o esforço e o ciclo de vida necessários para o desenvolvimento de determinadas aplicações distribuídas.

    - Os DOC *middlewares* povêm algumas funcionalidades para o desenvolvimento de aplicações distribuídas. Estas funcionalidades serão apresentadas a seguir, segundo Schantz [14].

- **Foco na integração ao invés da programação:** A origem do *middleware* se dá devido ao problema em relação à integração e construção de partes separadas de uma determinada aplicação. Middlewares distribuídos, a exemplo do CORBA e do Java RMI, possibilitam que pedaços de aplicações sejam interconectados, independentemente da localização e da tecnologia utilizada. Estas funcionalidades permitem que *middlewares* DOC reduzam os esforços do ciclo de vida do software por considerar experiências de desenvolvimento anteriores.
- **Demanda para suporte fim-a-fim de QoS, não apenas componente QoS:** A construção de aplicações que utilizam QoS não é uma tarefa fácil, visto que há uma necessidade de implementar as propriedades não funcionais do QoS, tais quais latência previsível, throughput, escalabilidade, dependência, flexibilidade e recursos integrados entre e dentro dos pedaços da aplicação. Existem duas premissas que vão de encontro ao suporte fim-a-fim do QoS, que são os diferentes níveis de serviços

possíveis e desejáveis, e o nível do serviço em uma propriedade deve ser coordenado para que o serviço desejado seja atingido.

- **A viabilidade crescente dos sistemas abertos:** a característica distribuída permite que os sistemas com este propósito sejam mais abertos que os sistemas monolíticos, facilitando a implementação de componentes a partir de uma múltipla escolha e do conceito de engenharia aberta.
- **Crescente demanda por tecnologias divididas que tendem ao aumento da competição global:** o custo do ciclo de vida do desenvolvimento de softwares pode ser amenizado pela consideração de conhecimento anterior e pela focalização no esforço para aumentar a qualidade do software. Quando os desenvolvedores não precisam se preocupar com os detalhes de baixo nível, eles podem se concentrar nos detalhes específicos da aplicação que estão desenvolvendo, como, por exemplo, gerenciamento de recursos distribuídos e dependência fim-a-fim.
- **Cobertura da complexidade potencial para sistemas complexos da próxima geração:** com a crescente complexidade dos sistemas distribuídos, fica difícil sustentar a integração e composição destes sem o uso contínuo de pesquisas. O que pode acontecer é a consideração de um conhecimento falho, levando a resultados de alto risco a problemas comuns.

CONCLUSÃO

Os objetivos principais de um *middleware* são a integração entre sistemas heterogêneos e a intermediação entre as aplicações e o sistema operacional. Para que estes objetivos sejam alcançados, um *middleware* deve fornecer serviços que atendam ao domínio de aplicações para o qual foi construído, sendo importante que este serviço tenha sua base em uma das API’s ou protocolos padrões.

A facilidade de uso dos *middlewares* é outro fator importante na sua elaboração, o que induz a criação de comandos intuitivos e aplicações com códigos fáceis de entender para se comunicar com a interface do *middleware* [3].

Muitos *middlewares* utilizam protocolos e API's proprietárias para o seu funcionamento, o que os limita para alguns ambientes. Códigos proprietários são necessários para resolver os problemas que não são endereçados pela padronização. Então, mesmo os vendedores que seguem a linha padrão necessitam colocar códigos proprietários para que os problemas que não estejam no domínio padrão possam ser considerados, permitindo também a concorrência entre os diversos vendedores [2].

Os componentes de um *middleware* estão se tornando mais importantes que os serviços oferecidos pelo sistema operacional (S.O), pois os primeiros estão substituindo as funções não distribuídas do S.O por funções distribuídas que utilizam redes. Por exemplo, as aplicações dependem mais no mecanismo de *Remote Procedure Call* (RPC) do que no nível de transporte de mensagens [4].

Uma desvantagem dos *middlewares* se concentra justamente em sua capacidade de amenizar a heterogeneidade, pois ele o faz adicionando uma falsa homogeneidade no sistema, o que apenas retarda a colisão entre os sistemas heterogêneos [2].

Os *middlewares* apresentam alguns desafios, que devem ser levados em consideração no momento da sua construção e utilização. Dentre eles pode-se destacar flexibilidade, performance, integração com a *Web* e entre *middlewares* e a computação [2].

Nos anos que se seguem, a tendência é a construção de "sistemas de sistemas", incluindo muitos níveis interdependentes, assim como múltiplas camadas de *middleware* comum e de domínio específico, interconexões de rede e de barramentos, etc. As características deste tipo de sistema incluem previsibilidade, controlabilidade e adaptabilidade de características operacionais [14].

Referências Bibliográficas e Créditos

[1] CAMPBELL, Andrew T.; COULSON, Geoff; KOUNAVIS, Michael E. . *Managing complexity: Middleware Explained.* IEEE, 1999, Setembro/Outubro. p. 22-28.

[2] VINOSKI, Steve. *Where is Middleware?* IEEE Internet Compuiting. Março/Abril. 2002. p. 83-85.

[3] CHARLES, John. *Middleware Moves to the Forefront*. Computer, Maio 1999. p. 17-19.

[4] BERNSTEIN, Philip A. *Middleware: A Model for Distributed System Services.* Communications of the ACM. Fevereiro, 1996. v. 39, nº 2. p. 86-98.

[5] BENDA, Miroslav. *Middleware: Any Client, Any Server*. IEEE Internet Computing. Julho/Agosto 1997. p. 94-96.

[6] TALARIAN. *Talarian: Everything you need to know about Middleware*. Disponível em: **http://www.talarian.com/industry/middleware/whitepaper.pdf** . Data de Acesso: 14/02/2003.

[7] GEIHS, Kurt. *Middleware Challenges Ahead*. Computer, Junho 2001. p. 24-30.

[8] EMMERICH, Wolfgang. *Software engineering and Middleware: A Roadmap.* Dept. Of Computer Science - University College London. 2000. Disponível em: **http://www.cs.ucl.ac.uk/staff/A.Finkelstein/fose/finalemmerich.pdf**.

[9] AGHA, Gul A. *Adaptive Middleware*. Communication of the ACM. Junho, 2002. v. 45. nº 6. pp. 31-32.

[10] ROMÁN, Manuel; KON, Fabio; CAMPBELL, Ray H. *Reflective Middleware: From Your Desk to your Hand.* Reflective Middleware. v. 2, nº 5, 2001. Disponível em: **http://www.ima.usp.br/~kon/papers/dsonline01.pdf** Data de Acesso: 19/02/2003.

[11] KON, Fábio et al. *The Case for Reflective Middleware*. Communications of the ACM. Junho 2001, v. 45, nº 6. p. 33-38.

[12] CAPRA, Licia; EMERICH, Wolfgang; MASCOLO, Cecília. *Reflective Middleware Solutions for Context-Aware Applications.* Dept. of Computer Science – University College London. Disponível em: **http://www.cs.ucl.ac.uk/staff/L.Capra/ref.pdf**. Data de Acesso: 21/02/2003.

[13] YAU, Stephen S.; Karim, Fariaz. *Adaptive Middleware for Ubiquitous Computing Environments*. Disponível em: Data de Acesso:

[14] SCHANTZ, Richard E.; SCHMIDT, Douglas C. *Middleware for Distributed Systems – Evolving the Commom Structure for Network-centric Applications* Encyclopedia of Software Engineering. 2001.

[15] ROSA, Nelson Souto. *Ambientes de Middleware*. Universidade Federal de Pernambuco. Disponível em: < **http://www.cin.ufpe.br/~sd/disciplinas/sd/pos/aulas/Middleware.pdf** >Acesso em:17/07/2003.

[16] OMG. CORBA. Disponível em: <**http://www.corba.org>**. Acesso em: 17/07/2003.

[17] SUN. *Java Remote Method Invocation* (RMI). Disponível em: <http://java.sun.com/products/jdk/rmi/>. Acesso em: 17/07/2003.

[18] W3C. *Simple Object Access Protocol* (SOAP) 1.1. Disponível em: <http://www.w3.org/TR/SOAP/> . Acesso em: 17/07/2003.

[19] MICROSOFT. *DCOM*. Diposnível em: <http://www.microsoft.com/com/tech/dcom.asp>. Acesso em: 17/07/2003.