Abordagens de utilização de arquiteturas *middleware* em aplicações robóticas embarcadas

Gustavo André Nunes Ferreira
Doutorando em Engenharia Mecatrônica e
Mestre em Engenharia – Poli-USP;
Professor na graduação [Ciência da Computação] – Uninove.
gandre@uninove.br, São Paulo – SP [Brasil]

Sistemas embarcados, em especial os robóticos, apresentam, em sua estrutura, uma multiplicidade de dispositivos que – com base, não raro, numa variedade tecnológica tanto em *hardware* quanto em *software* e conectada com diferentes tecnologias e protocolos de redes – resulta em uma arquitetura bastante heterogênea e bem distribuída. Para auxiliar no enfrentamento dessa complexidade inerente aos sistemas distribuídos, foi desenvolvido o conceito de arquitetura *middleware*, que consiste em uma camada de integração de *software*. O objetivo central deste artigo é expor, por meio de uma visão geral, a utilização de arquiteturas *middleware* em aplicações de robótica embarcada, especialmente as que possuem dispositivos de escassa disponibilidade de recursos (memória, capacidade de processamento e comunicação), como microcontroladores e sensores inteligentes (em inglês *smart sensors*), além de analisar diferentes tipos de abordagem para que essas arquiteturas sejam usadas adequadamente.

Palavras-chave: Arquiteturas *middleware*. Sistemas embarcados. Sistemas robóticos.

1 Introdução

Sistemas embarcados, particularmente os robóticos, são constituídos de múltiplos dispositivos de diferentes naturezas. Invariavelmente, essas aplicações possuem três conjuntos de elementos básicos: sensores (de deslocamento, posicionamento etc.), para a percepção de seu ambiente de execução; atuadores, representados tipicamente por motores; e uma ou mais unidades de processamento das informações do sistema (do mais simples microprocessador até os mais sofisticados microcomputadores e microcontroladores).

A arquitetura *middleware* consiste, basicamente, em uma camada de integração de *software*, residente acima do sistema operacional e do substrato de comunicação, que oferece abstrações de alto nível e um conjunto de serviços, com o objetivo de facilitar a programação distribuída. Atualmente, são considerados como exemplos de arquiteturas *middleware* bem conhecidas: *remote procedure call* (RPC), modelo de programação cliente/servidor originalmente da Sun; *distributed component object model* (Dcom), modelo de componentes distribuídos da Microsoft destinado à plataforma Windows; Java *remote method invocation* (RMI), modelo de invocação remota da comunidade Java; *common object request broker architecture* (Corba), padronização da *Object* Management Group (OMG) para computação de objetos distribuídos, independentemente de plataforma e da linguagem de programação adotadas (GILL; SMART, 2002).

Há um número considerável de publicações que trata da utilização de arquiteturas *middleware*, particularmente, de implementações Corba, em aplicações robóticas, tais como: Bottazzi e outros (2002), em sistemas de teleoperação robótica; Utz e outros (2002), em aplicações de robôs móveis; Schmidt (2002) e Gong (2003), em sistemas de tempo real e embarcados; Sanz e Alonso (2001), que tecem considerações sobre o uso de arquiteturas Corba em sistemas de controle, citando aplicações em teleoperação robótica, em sistemas de gerenciamento de risco, em processos de química industrial, em sistemas de transmissão de vídeo em tempo real.

Este trabalho tem como finalidade apresentar uma visão geral do uso de arquiteturas *middleware* em aplicações de robótica embarcada[1], especialmente, as que possuem dispositivos de escassa disponibilidade de recursos (memória, capacidade de processamento e comunicação), tais como microcontroladores e sensores inteligentes (em inglês, *smart sensors*). Este estudo busca analisar os diferentes tipos de abordagens existentes para sua conveniente utilização, de acordo com os dispositivos e as aplicações.

2 Arquiteturas *middleware* para aplicações embarcadas

Segundo a literatura especializada, um dos principais argumentos contra a utilização de arquiteturas *middleware* em aplicações embarcadas, sejam robóticas ou gerais, é que elas acrescentam um código desnecessário ao sistema, prejudicando o seu desempenho. O argumento torna-se ainda mais forte quando se trata de aplicações de escassos recursos de memória, processamento e comunicação. Implementações de arquiteturas *middleware* de uso geral podem, de fato, ter uma expressiva quantidade de código não útil, especificamente, no sistema sob o qual estão sendo utilizadas. Por outro lado, apenas um subconjunto de tais arquiteturas pode oferecer muitos dos benefícios, advindos do uso dessas estruturas, sem, no entanto, prejudicar, significantemente, o desempenho do sistema.

Recentes pesquisas nessa área têm identificado diversas formas que asseguram somente as carac-

terísticas necessárias da arquitetura *middleware* para a aplicação alvo, que provoca uma redução em termos de impacto no desempenho caso a caso. Nesse contexto, as formas de enfrentamento atual dessa questão, podem ser divididas em três abordagens principais: a primeira com base na utilização de delegação; a segunda, fundamentada na redução da infra-estrutura *middleware*; e a terceira, na utilização de arquiteturas constituídas por componentes.

2.1 Abordagem via delegação

Nesta abordagem, pressupõe-se a presença de um sistema com recursos para a utilização de uma infra-estrutura *middleware* tradicional, com a qual se deseja integrar um ou mais dispositivos que não apresentam recursos para suportar essa infra-estrutura. A expectativa é de que eles disponibilizem algum tipo de serviço para o sistema como um todo. Como solução para o cenário anteriormente descrito, esta abordagem faz com que os dispositivos que não suportam a infra-estrutura *middleware* utilizem algum tipo de delegação (tipicamente um *proxy*) para se fazerem presentes nela, disponibilizando, desta forma, seus serviços.

Pode-se citar, como exemplos, nesse contexto, a tecnologia Jini sobre RMI (SUN MICROSYSTEMS, 2003) da Sun, e a proposta *smart transducers interface* sobre Corba (OBJECT MANAGEMENT GROUP, 2003) da OMG.

2.2 Abordagem via redução da infra-estrutura *middleware*

Em geral, as bibliotecas que implementam uma arquitetura *middleware* tradicional exigem vários megabytes de memória, o que está além da capacidade dos microcontroladores tipicamente utilizados em aplicações embarcadas.

Nesta abordagem, procura-se reduzir a infra-estrutura do *object request broker* (ORB) da arquitetura *middleware* carregada no dispositivo embarcado, pela omissão de funcionalidades não interessantes para a aplicação, mantendo, porém, a interoperabilidade com a arquitetura *middleware* original. Dessa forma, tal infra-estrutura exigirá pouco mais de uma centena de Kbytes de memória, podendo ser carregada por microcontroladores típicos de aplicações embarcadas.

A especificação Minimum Corba (OBJECT MANAGEMENT GROUP, 2002) é um exemplo da abordagem baseada na redução da infra-estrutura *middleware*.

2.3 Abordagem, via *middleware*, com base em componentes

Implementações tradicionais de arquiteturas *middleware* são monolíticas, ou seja, todo o mecanismo interno do *middleware* é apresentado como uma caixa-preta e estará no sistema, independentemente do usuário querer ou não. Mesmo que a aplicação necessite apenas de 10% desse mecanismo, o que é comum para a maioria das aplicações reais, o código completo é carregado nela.

Sistemas com base em componentes são formados por meio da combinação de vários componentes de *software*, normalmente armazenados no sistema de arquivos local ou em um repositório acessível na rede. Nesta abordagem, programadores determinam quais desses elementos farão parte do sistema e como será o inter-relacionamento entre eles e especificam os serviços do ORB que serão necessários e que serão carregados pelo *middleware* em tempo de inicialização. Trata-se da mais flexível das três abordagens apresentadas neste artigo e a que oferece mais vantagens no confronto das complexidades advindas da utilização de arquiteturas *middleware* em sistemas embarcados.

Um exemplo de *middleware*, com base em componentes, é o *universally interoperable core*

(UIC) (KON et al., 2002), desenvolvido pelo Ubicore LLC (http://www.ubi-core.com).

3 Exemplos de abordagens *middleware* para sistemas embarcados

Conforme enumerações a seguir.

3.1 Jini sobre RMI

A tecnologia Jini surgiu com a proposta de tornar a computação distribuída uma atividade mais fácil, para permitir que dispositivos criem uma rede de comunicação dinâmica e nela compartilhem serviços, formando uma *federation* com base na arquitetura *middleware* RMI. Cada dispositivo expõe serviços que outros dispositivos da *federation* podem utilizar.

Para o caso de dispositivos com restrições de recursos que os impossibilitem de suportar o RMI, a tecnologia Jini propõe dois tipos de solução. Na primeira, um grupo de dispositivos usa uma máquina virtual Java fisicamente compartilhada como camada intermediária entre um dispositivo e o sistema ou entre dispositivos utilizando a tecnologia Jini (SUN MICROSYSTEMS, 2003) (Ilustração 1). Cada dispositivo carrega o código escrito em linguagem de programação Java na máquina virtual, o que permite que ela interaja com o dispositivo e delegue para a máquina virtual as requisições de interação do sistema.

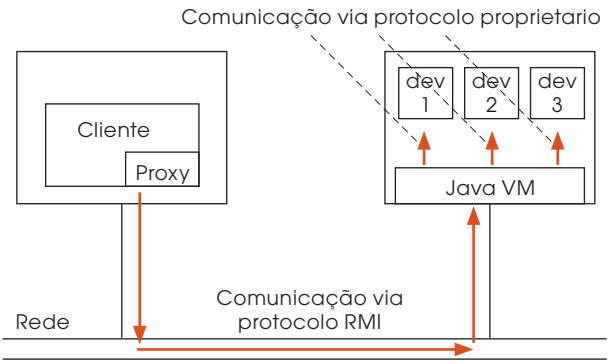

Ilustração 1: Tecnologia Jini sobre RMI: JVM fisicamente compartilhada

Fonte: O autor.

Já no segundo tipo de solução, existirá na rede um *proxy* da máquina virtual Java compartilhada com vários dispositivos que, ao serem adicionados à rede, podem descobrir a existência de tal *proxy* e, em seguida, utilizá-lo para registrar-se. O registro pode incluir o código escrito em linguagem de programação Java, o que é necessário para um cliente do dispositivo, além do código fundamental para que haja a comunicação entre o *proxy* e o dispositivo (SUN MICROSYSTEMS, 2003) (Ilustração 2).

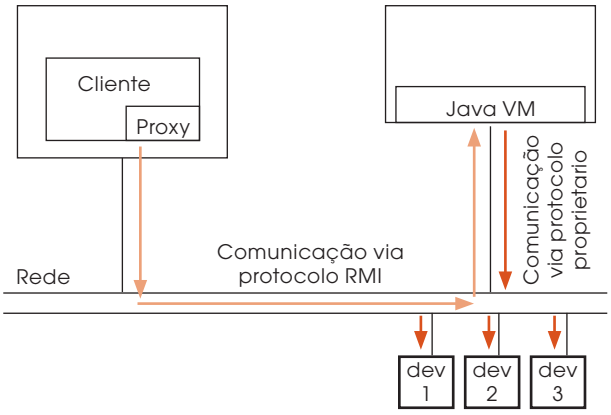

Ilustração 2: Tecnologia Jini sobre RMI: JVM compartilhada via rede

Fonte: O autor.

Um exemplo de uso da tecnologia Jini em aplicações robóticas embarcadas é citado em Long e outros (2005), que apresentam a *distributed field robot architecture* (DFRA), uma implementação de arquitetura de gerenciamento orientada para objetos distribuídos. Ela provê acesso consistente aos recursos (tais como sensores, atuadores e algoritmos) de um time de robôs por meio de serviços Jini, de forma que eles possam ser dinamicamente descobertos e utilizados pelos robôs ou por operadores humanos.

3.2 *Smart transducers interface* sobre Corba

Diferentemente do Jini sobre RMI, a proposta de padronização *smart transducers interface* da OMG não restringe a linguagem de programação utilizada, oferecendo ao programador transparência adicional em relação à Jini sobre RMI.

Essa especificação permite que dispositivos sejam endereçados como se estivessem inseridos nos sistemas que suportam um ORB Corba completo (GILL; SMART, 2002). Para isto, uma aplicação local de um *smart transducer* deve mapear as funcionalidades do dispositivo em um sistema de arquivo de *interface* (*interface file system* [IFS]) que compreende um conjunto de pequenos elementos de memória que são tipicamente alocados na memória local do dispositivo. O IFS é hierarquicamente estruturado nos arquivos e registros, o que resulta em um esquema de endereçamento único. Tais arquivos e registros são facilmente traduzidos em vários protocolos, que possibilitam que haja a interoperabilidade de diversos sistemas (Ilustração 3). Um *smart transducer* comum caberá em um microcontrolador do tipo *reduced instruction set computer* (Risc ou, em português, conjunto reduzido de instruções computacionais) de 8 bits com 128 bytes de RAM e 4 Kbytes de ROM (ELMENREICH; PITZEK, 2003).

Elmenreich e Obermaisser (2002) apresentam a implementação de dois casos de estudo da *smart transducers interface* em sistemas robóticos. O primeiro consiste em uma demonstração feita com um manipulador robótico que é instrumentado por uma rede de *smart transducers* divididos em dois grupos, que possuem sensores, atuadores e unidades de comando. O segundo caso utiliza-se de um robô móvel autônomo para mostrar a integração de novos recursos e a comunicação eficiente entre eles por meio de *smart transducers interface*, o que comprova que, por meio da tecnologia,

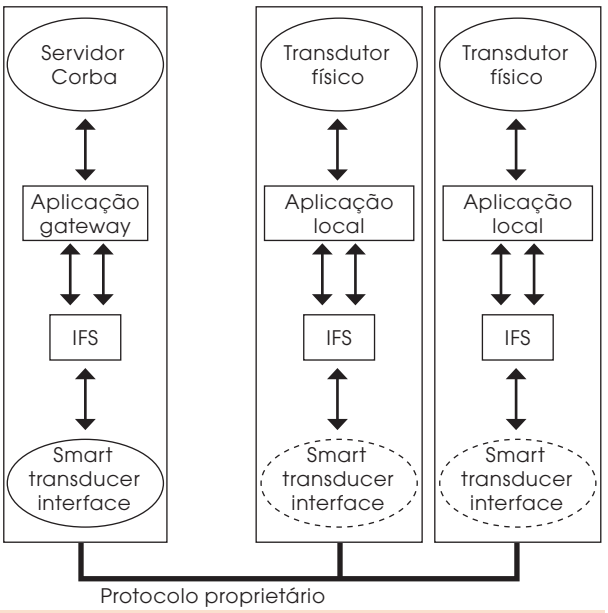

Ilustração 3: Exemplo de aplicação utilizando *smart transducers interface* sobre Corba

Fonte: O autor.

há a possibilidade tanto de implantar neles essas características tão complexas, tais como *plug & play*, quanto de reconfigurá-los.

3.3 Minimum Corba

A especificação Minimum Corba é um subconjunto projetado para sistemas com recursos limitados que omite algumas características da especificação original, que ainda podem ser implementadas pela aplicação, caso elas sejam necessárias (Ilustração 4). Entre as características omitidas, destacam-se os aspectos dinâmicos da arquitetura Corba (*dynamic invocation interface* [DDI] e *dynamic skeleton interface* [DSI]), com a maior parte do Repositório de Interfaces e o suporte a interceptadores. Além disso, diversas funcionalidades não essenciais são omitidas das interfaces do ORB e do *portable object adapter* (POA). Apesar de todos esses cortes, o Minimum Corba preserva a interoperabilidade, o que implica o suporte à especificação completa de Corba IDL (em português, Linguagem de Definição de Interfaces de Corba).

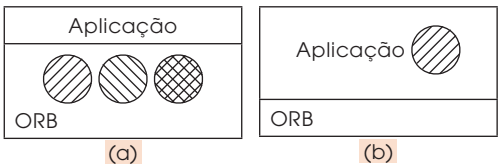

Ilustração 4: Características, representadas por círculos hachurados, omitidas no Minimum Corba

Obs.: 4a = Corba; 4b = Minimum Corba.

Fonte: O autor.

As Ilustrações 5 e 6 apresentam, a título de comparação, diagramas esquemáticos representando os componentes do ORB Corba e do ORB Minimum Corba, respectivamente.

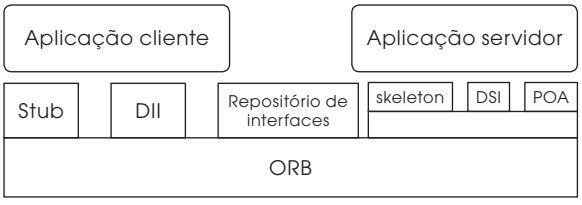

Ilustração 5: Componentes do ORB Corba

Fonte: O autor.

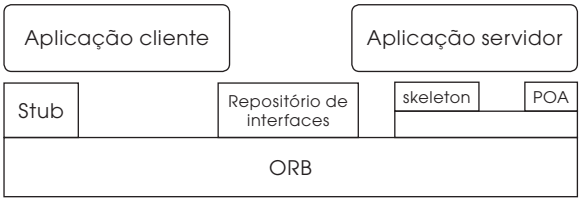

Ilustração 6: Componentes do ORB Minimum Corba

Fonte: O autor.

Schmidt (2002) cita o ACE ORB (TAO), uma implementação de código aberto do Corba em C++, que também inclui uma especificação Minimum Corba que é amplamente utilizada na arquitetura Miro (UTZ et al., 2002), um *framework* estruturado em camadas arquiteturais que é usado, com sucesso, em vários projetos destinados a aplicações de robôs móveis autônomos, tais como mapeamento de ambientes internos, modelagem de ambiente de execução, localização, planejamento de trajetória, entre outros.

3.4 *Universally interoperable core*

O UIC define um esqueleto com base em componentes abstratos que encapsulam as funcionalidades encontradas nos principais ORBs existentes. Implementações concretas desses componentes abstratos são carregadas dinamicamente de forma a atender aos requisitos de aplicações específicas.

UIC determina diferentes personalidades (instâncias particulares de um UIC obtido por especialização) atendendo às especificações de diferentes padrões de *middleware*. A Ilustração 7 apresenta um diagrama esquemático representando o UIC abstrato (UIC), suas especializações (UIC-Corba e UIC-RMI) e os servidores que exportam funcionalidades delas (Servidor Corba e Servidor RMI).

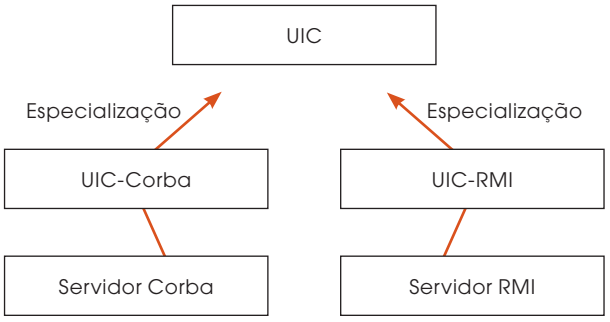

Ilustração 7: Diagrama esquemático do UIC

Fonte: O autor.

No momento, a única personalidade em estágio avançado de desenvolvimento é UIC-Corba, capaz de interagir com ORBs Corba utilizando muito menos recursos do que ORBs tradicionais.

Segundo Costa e Kon (2002), a vantagem de ORBs, com base em componentes para sistemas embarcados, pode ser ilustrada pelo tamanho do código das bibliotecas implementando um

cliente mínimo em UIC-Corba. O tamanho do *middleware* para um cliente mínimo com capacidade de enviar requisições simples para um servidor Corba padrão é de 29 Kbytes no Windows CE, 72 Kbytes no Windows 2000 e 18 Kbytes no PalmOS.

Em razão desta abordagem ser a mais nova comparativamente às outras mencionadas neste trabalho, não há relatos da utilização de ORBs, com base em componentes, na implementação de aplicações robóticas. O foco dos desenvolvimentos dessas ORBs inclui o UIC e tem como alvo, principalmente, as aplicações de computação móvel e ubíqua.

4 Análises e conclusões

Este artigo é o resultado de uma pesquisa bibliográfica a respeito dos diferentes tipos de abordagem que são necessárias para se enfrentar as complexidades e as heterogeneidades inerentes aos sistemas embarcados por meio de arquiteturas *middleware*. Particularmente, enfocaram-se sistemas com limitada disponibilidade de recursos (memória, capacidade de processamento e comunicação), visando a aplicações robóticas.

Com base nos exemplos encontrados, observou-se que há uma tendência de se escolher as implementações Corba. Esse fato se justifica, como citado por Gill e Smart (2002), por dois fatores principais em relação ao Corba: por constituir um padrão aberto para uso e extensão e fechado para modificações, sem completa revisão sob o processo de padronização de OMG; por ser independente de plataforma e linguagem de programação.

A primeira abordagem *middleware*, citada neste artigo, para defrontar os problemas em foco foi a baseada em delegação, que figura como uma opção que traz vantagens em casos específicos, particularmente, naqueles em que se deseja disponibilizar funcionalidades de um conjunto de dispositivos para um sistema que suporta uma infra-estrutura *middleware* completa.

Em relação à abordagem com base na redução da infra-estrutura *middleware*, explorou-se, como exemplo, a especificação Minimum Corba (OBJECT MANAGEMENT GROUP, 2002). Ela se fundamenta na omissão de uma série de características do Corba original, em especial as características dinâmicas, cabendo aqui uma observação: recentes pesquisas, entre elas Kristensen e outros (2003). Kon e outros (2002), Costa e Kon (2002); Román, Kon e Campbell (2001) e Eliassen e outros (1999), que têm mostrado o fecundo campo que há para a utilização de arquiteturas *middleware* reflexivas[2] em sistemas embarcados, especialmente em computação ubíqua. Tais arquiteturas são essencialmente dinâmicas, o que desprestigiaria a iniciativa Minimum Corba. Nesse contexto, a abordagem que se apresenta como a mais promissora, tanto para o desenvolvimento de aplicações embarcadas tradicionais quanto para as aplicações em computação ubíqua, é a que utiliza arquiteturas *middleware* baseadas em componentes.

Como possíveis trabalhos de pesquisa que podem ser desenvolvidos por meio dos estudos realizados para a elaboração deste artigo, citam-se: um melhor aprofundamento a respeito das arquiteturas *middleware*, baseadas em componentes e das arquiteturas *middleware* reflexivas; uma investigação sobre o impacto que a utilização de arquiteturas *middleware* gera no desempenho de sistemas embarcados reais; um trabalho a respeito dos reais ganhos (em termos do tempo de desenvolvimento, reusabilidade, escalabilidade etc.) advindos do uso de arquiteturas *middleware* no desenvolvimento de sistemas embarcados.

Approaches of the use of middleware architecture in embedded robotic applications

Embedded systems, specially the robotic ones, present a multiplicity of devices in its structure which – often based on different technologies of hardware and software and connected by different network technologies and protocols –, results in a considerably heterogeneous and well distributed architecture. In order to improve the comprehension over the complexity inherent to the distributed systems the new concept of middleware emerged as a software integration layer. The main objective of this article is to expose, by means of a general view, the use of middleware architectures in embedded robotic applications, particularly that with devices of scarce availability of resources (memory, processing capacity and communication), such as microcontrollers and smart sensors, and to analyze different approaches for a convenient use of these architectures.

Key words: Embedded systems. Middleware architecture. Robotic systems.

Notas

1 Este trabalho limita-se às arquiteturas *middleware* de especificação aberta, particularmente as relacionadas às especificações Corba e J2EE.

2 Ubiqüidade refere-se à capacidade de um determinado dispositivo móvel estar constantemente conectado a uma rede, acessível sob as mais variadas situações.

Referências

BOTTAZZI, S. et al. A software framework based on real-time Corba for Telerobotic Systems. In: IEEE INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS, Lausanne. *Proceedings*... Lausanne: Iros, 2002.

COSTA, F. M.; KON, F. Novas tecnologias de middleware: Rumo à flexibilização e ao dinamismo In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 20., Búzios. *Anais*... Búzios: SBC, 2002. p. 1-61.

ELIASSEN, F. et al. Next generation middleware: Requirements, architecture, and prototypes. In: IEEE WORKSHOP ON FUTURE TRENDS OF *DISTRIBUTED* COMPUTING SYSTEMS, 17., Tunísia. *Proceedings*... Tunísia: IEEE Computer Society, 1999. p. 60-65.

ELMENREICH, W.; OBERMAISSER, R. A standardized smart transducer interface. In: IEEE INTERNATIONAL SYMPOSIUM ON INDUSTRIAL ELECTRONICS. L'Aquila. *Proceedings*... L'Aquila: IEEE Industrial Electronics Society, 2002. p. 164-169.

_____.; PITZEK, S. Smart transducers: principles, communications and configuration. In: IEEE International Conference on Intelligent Engineering Systems, 7., Luxor. *Proceedings*... Luxor: Ines, 2003. p. 510-515.

GILL, C. D.; SMART, W. D. Middleware for robots? In: AAAI SPRING SYMPOSIUM ON INTELLIGENT DISTRIBUTED AND EMBEDDED SYSTEMS. Stanford. *Proceedings*... Stanford: 2002.

KON, F. et al. The case for reflective middleware. *Communications of the ACM*, Nova York, v. 45, n. 6, p. 33-38. Disponível em: <http://portal.acm.org/ft_gateway.cfm?id=508470&type=pdf&coll=GUIDE&dl=GUIDE&CFID=15151515&CFTOKEN=6184618>. Acesso em: 18 fev. 2006.

KRISTENSEN, T. et al. Interactive multimedia on next generation networks. Evaluation of middleware for distributed objects on handheld devices. *Lecture Notes in Computer Science*, Heidelberg, v. 2.899, p. 270-281, 2003.

LONG, M. et al. Application of the distributed field robot architecture to a simulated demining task. In: IEEE International Conference on Robotics and Automation. Barcelona. *Proceedings*... Barcelona: IEEE, 2005.

OBJECT MANAGEMENT GROUP. *Minimum Corba specification. Version 1.0. Technical Report formal/02-08-01*, Needham, ago. 2002. Disponível em: <http://www.omg.org/docs/formal/02-08-01.pdf>. Acesso em: 18 fev. 2006.

OBJECT MANAGEMENT GROUP. *Smart transducers interface specification. Version 1.0. Technical Report formal/03-01-01*. Needham: Object Management Group, 2003. Disponível em: <http://www.omg.org/docs/formal/03-01-01.pdf>. Acesso em: 18 fev. 2006.

ROMÁN, M.; KON, F.; CAMPBELL, R. H. Reflective middleware: from your desk to your hand. *IEEE Distributed Systems Online*, Washington, v. 2, n. 5, 2001. Disponível em: <http://csdl.computer.org/comp/mags/ds/2001/05/o5001.pdf>. Acesso em: 18 fev. 2006.

SANZ, R.; ALONSO, M. Corba for control systems. *Annual Reviews in Control*, Luxemburgo, v. 25, n. 0, p. 169-181, 2001.

SCHMIDT, D. C. Middleware for real-time and embedded systems. *Communications of the ACM*, Nova York, v. 45, n. 6, p. 43-48, 2002. Disponível em: <http://portal.acm.org/ft_gateway.cfm?id=508472&type=pdf&coll=portal&dl=ACM&CFID=15151515&CFTOKEN=6184618>. Acesso em: 18 fev. 2006.

SUN MICROSYSTEMS. *Jini network technology*. Santa Clara: Sun MicroSystems, 2003. Disponível em: <http://www.sun.com/jini>. Acesso em: 18 fev. 2006.

UTZ, H. et al. Miro: middleware for mobile robot applications. *IEEE Transactions on Robotics and Automation*, Nova York, v. 18, n. 4, p. 493-497, 2002.

Recebido em: 18 fev. 2006 / aprovado em: 7 maio 2006

Para referenciar este texto

FERREIRA, G. A. N. Abordagens de utilização de arquiteturas middleware em aplicações robóticas embarcadas. *Exacta*, São Paulo, v. 4, n. 1, p. 149-157, jan./jun. 2006.