# 8

# Referências Bibliográficas


[1] LEHMAN, M. M.. **Laws of software evolution revisited**. Em: Montangero, C., editor, SOFTWARE PROCESS TECHNOLOGY: $5^{TH}$ EUROPEAN WORKSHOP, EWSPT '96, NANCY, FRANCE, OCTOBER 9–11, 1996: PROCEEDINGS, volume 1149 de **Lecture Notes in Computer Science**, p. 108–124, Nanci, França, outubro 1996. IEEE, Springer-Verlag Heidelberg. ISBN: 3-540-61771-X.

[2] STAA, A. V.. **Programação modular: desenvolvendo programas complexos de forma organizada e segura**. Campus, Rio de Janeiro, Brasil, abril 2000.

[3] GAMMA, E.; HELM, R.; JOHNSON, R. ; VLISSIDES, J.. **Design Patterns — Elements of Reusable Object-Oriented Software**. Addison-Wesley Professional, Boston, EUA, outubro 1994. ISBN: 0-201-63361-2.

[4] SZYPERSKI, C.. **Component Software: Beyond Object-Oriented Programming**. Component Software. Addison-Wesley Professional, Boston, EUA, segunda edição, novembro 2002. ISBN: 0-201-74572-0.

[5] Liu, X.; Yang, H., editors. **Principles of Software Evolution (ISPSE 2000) (Postproceedings): 2000 International Symposium**, Kanazawa, Japão, novembro 2000. IEEE Computer Society, IEEE Press. ISBN: 0-769-50906-1.

[6] STEIN, L. A.; LIEBERMAN, H. ; UNGAR, D.. **A shared view of sharing: The Treaty of Orlando**. Em: Kim, W.; Lochovsky, F. H., editors, OBJECT-ORIENTED CONCEPTS, DATABASES AND APPLICATIONS, p. 31–48. ACM Press/Addison-Wesley, Boston, EUA, 1989.

[7] PFISTER, C.; SZYPERSKI, C.. **Why objects are not enough**. Em: Jell, T., editor, COMPONENT-BASED SOFTWARE ENGINEERING: $1^{ST}$ INTERNATIONAL COMPONENT USERS CONFERENCE CUC'96, MUNICH, GERMANY, PROCEEDINGS, p. 165, Munique, Alemanha, julho 1996. Cambridge University Press/SIGS Books.

[8] RADENSKI, A.. **Anomaly-free component adaptation with class overriding**. Journal of Systems and Software, 71(1–2):37–48, abril 2004. Elsevier Science.

[9] UDELL, J.. **Componentware**. Byte, 19(5):46–56, maio 1994.

[10] KICZALES, G.; LAMPING, J.; MENHDHEKAR, A.; MAEDA, C.; LOPES, C.; LOINGTIER, J.-M. ; IRWIN, J.. **Aspect-oriented programming**. Em: Akşit, M.; Matsuoka, S., editors, ECOOP '97 - OBJECT-ORIENTED PROGRAMMING: $11^{TH}$ EUROPEAN CONFERENCE, JYVÄSKYLÄ, FINLAND, JUNE 9–13, 1997: PROCEEDINGS, volume 1241 de **Lecture Notes in Computer Science**, p. 220–242, Jyväskylä, Finlândia, junho 1997. Association Internationale pour les Technologies Objets, Springer-Verlag Heidelberg.

[11] Object Management Group, Needham, EUA. **CORBA Component Model - Version 3.0**, junho 2002. document: formal/2002-06-65.

[12] MCINROY, D.. **Mass produced software components**. Em: Naur, P.; Randell, B., editors, SOFTWARE ENGINEERING, p. 138–155. NATO Science Comitee Report, 1968.

[13] BRUNETON, E.; COUPAYE, T. ; STEFANI, J.. **Recursive and dynamic software composition with sharing**. Em: Núñez, J. H.; Moreira, A. M. D., editors, OBJECT-ORIENTED TECHNOLOGY. ECOOP 2002 WORKSHOP READER : ECOOP 2002 WORKSHOPS AND POSTERS, MÁLAGA, SPAIN, JUNE 10–14, 2002. PROCEEDINGS, volume 2548 de **Lecture Notes in Computer Science**, Málaga, Espanha, junho 2002. Association Internationale pour les Technologies Objets, Springer-Verlag Heidelberg.

[14] PESCHANSKI, F.; BRIOT, J.-P. ; YONEZAWA, A.. **Fine-grained dynamic adaptation of distributed components**. Em: Endler, M.; Schmidt, D., editors, MIDDLEWARE 2003: ACM/IFIP/USENIX INTERNATIONAL MIDDLEWARE CONFERENCE, RIO DE JANEIRO, BRAZIL, JUNE 16–20, 2003, PROCEEDINGS, volume 2672 de **Lecture Notes in Computer Science**, p. 123–142, Rio de Janeiro, Brasil, junho 2003. ACM/IFIP/USENIX, Springer-Verlag Heidelberg. ISBN: 3-540-40317-5.

[15] FLATT, M.. **Programming Languages for Reusable Software Components**. PhD thesis, Rice University, Houston, EUA, junho 1999. TR99-345, 158 p.

[16] Object Management Group, Needham, EUA. **Common Object Request Broker Architecture: Core Specification - Version 3.0**, dezembro 2002. document: formal/2002-12-06.

[17] WANG, N.; SCHMIDT, D. C. ; O'RYAN, C.. **An overview of the CORBA component model**. Em: Heineman, G.; Councill, B., editors, COMPONENT-BASED SOFTWARE ENGINEERING. Addison-Wesley Professional, Boston, EUA, 2000.

[18] Robertson, P.; Shrobe, H. ; Laddaga, R., editors. **Self-Adaptive Software. First International Workshop, IWSAS 2000 Oxford, UK,**

**April 17-19, 2000 Revised Papers**, volume 1936 de **Lecture Notes in Computer Science**, Berlin, Alemanha, junho 2001. Springer-Verlag Heidelberg. ISBN: 3-540-41655-2.

[19] DEMERS, F.-N.; MALENFANT, J.. **Reflection in logic, functional and object-oriented programming: a short comparative study**. Em: IJCAI'95 WORKSHOP ON REFLECTION AND METALEVEL ARCHITECTURES AND THEIR APPLICATIONS IN AI, p. 29–38, Montreal, Canadá, agosto 1995. IJCAII/AAAI/CSCSI, Morgan Kaufmann.

[20] MOURA, A. L.; URURAHY, C.; CERQUEIRA, R. F. G. ; RODRIGUEZ, N. R.. **Dynamic support for distributed auto-adaptive applications**. Em: Wagner, R., editor, $22^{ND}$ INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS WORKSHOPS: 2–5 JULY 2002 VIENNA, AUSTRIA: PROCEEDINGS, p. 451–458, Viena, Áustria, julho 2002. IEEE Computer Society, IEEE Press. ISBN: 0-769-51588-6.

[21] SILVA, F. J. S.. **Adaptação Dinâmica de Sistemas Distribuídos**. Tese de Doutorado, Instituto de Matemática e Estatística — Universidade de São Paulo, São Paulo, Brasil, janeiro 2003.

[22] DOWLING, J.; SCHAFER, T.; CAHILL, V.; HARASZTI, P. ; REDMOND, B.. **Using reflection to support dynamic adaptation of system software: A case study driven evaluation**. Em: Cazzola, W.; Stroud, R. J. ; Tisato, F., editors, REFLECTION AND SOFTWARE ENGINEERING, PAPERS FROM OORaSE 1999, $1^{ST}$ OOPSLA WORKSHOP ON REFLECTION AND SOFTWARE ENGINEERING, volume 1826 de **Lecture Notes in Computer Science**, p. 169–188, Denver, EUA, novembro 1999. Springer-Verlag Heidelberg. ISBN: 3-540-67761-5.

[23] SZYPERSKI, C.. **Greetings from DLL hell**. Software Development, 7(10), outubro 1999. Beyond Objects column.

[24] ANDRADE, L. F.; FIADEIRO, J. L.. **Coordination: The evolutionary dimension**. Em: Pree, W., editor, $38^{TH}$ TECHNOLOGY OF OBJECT-ORIENTED LANGUAGES AND SYSTEMS (TOOLS EUROPE 2001): COMPONENTS FOR MOBILE COMPUTING, p. 136–147, Zurique, Suíça, março 2001. Interactive Software Engineering, IEEE Press. ISBN: 0-769-51095-7.

[25] GOUVEIA, J.; KOUTSOUKOS, G.; ANDRADE, L. ; FIADEIRO, J.. **Tool support for coordination-based software evolution**. Em: Pree, W., editor, $38^{th}$ TECHNOLOGY OF OBJECT-ORIENTED LANGUAGES AND SYSTEMS (TOOLS EUROPE 2001): COMPONENTS FOR MOBILE COMPUTING, p. 184–196, Zurique, Suíça, março 2001. Interactive Software Engineering, IEEE Press. ISBN: 0-769-51095-7.

[26] PAPADOPOULOS, G. A.; ARBAB, F.. **Coordination models and languages**. Em: Zelkowitz, M. V., editor, THE ENGINEERING OF LARGE SYSTEMS, volume 46 de **Advances in Computers**, p. 329–400. Academic Press, Finlândia, 1998.

[27] VOELTER, M.. **Aspectj-oriented programming in Java**. Java Report, janeiro 2000.

[28] KON, F.; COSTA, F.; BLAIR, G. ; CAMPBELL, R. H.. **The case for reflective middleware**. Communications of ACM, 45(6):33–38, junho 2002.

[29] BATISTA, T.; CERQUEIRA, R. F. G. ; RODRIGUEZ, N. R.. **Enabling reflection and reconfiguration in CORBA**. Em: MIDDLEWARE2003 COMPANION — THE $2^{ND}$ WORKSHOP ON REFLECTIVE AND ADAPTIVE MIDDLEWARE (EM CONJUNTO COM A ACM/IFIP/USENIX INTERNATIONAL MIDDLEWARE CONFERENCE – MIDDLEWARE'2003), p. 125–129, Rio de Janeiro, Brasil, junho 2003. ACM/IFIP/USENIX. ISBN: 8-587-92603-9.

[30] KON, F.; ROMÁN, M.; LIU, P.; MAO, J.; YAMANE, T.; MAGALHÃES, L. C. ; CAMPBELL, R. H.. **Monitoring, security, and dynamic configuration with the dynamicTAO reflective ORB**. Em: Sventek, J.; Coulson, G., editors, IFIP/ACM INTERNATIONAL CONFERENCE ON DISTRIBUTED SYSTEMS PLATFORMS AND OPEN DISTRIBUTED PROCESSING NEW YORK, NY, USA, APRIL 4-7, 2000 PROCEEDINGS, volume 1795 de **Lecture Notes in Computer Science**, p. 121–143, Nova Iorque, EUA, abril 2000. IFIP/ACM, Springer-Verlag Heidelberg.

[31] TRUYEN, E.; JØRGENSEN, B. N. ; JOOSEN, W.. **Customization of component-based object request brokers through dynamic configuration**. Em: $37^{TH}$ TECHNOLOGY OF OBJECT-ORIENTED LANGUAGES AND SYSTEMS (TOOLS EUROPE 2000), p. 181–194, Normandia, França, junho 2000. Interactive Software Engineering, IEEE Press. ISBN: 0-769-50731-X.

[32] CERQUEIRA, R. F. G.; CASSINO, C. ; IERUSALIMSCHY, R.. **Dynamic component gluing across different componentware systems**. Em: 1999 INTERNATIONAL SYMPOSIUM ON DISTRIBUTED OBJECTS AND APPLICATIONS (DOA '99), p. 362–373, Edinburgo, Escócia, setembro 1999. IEEE Press. ISBN: 0-769-50182-6.

[33] IERUSALIMSCHY, R.; FIGUEIREDO, L. H. ; FILHO, W. C.. **Lua — an extensible extension language**. Software: Practice and Experience, 26(6):635–652, junho 1996.

[34] CERQUEIRA, R. F. G.; NOGUEIRA, L. ; MOURA, A.. **The LuaOrb Manual**. TeCGraf — PUC-Rio, Rio de Janeiro, Brasil, novembro 2000. http://www.tecgraf.puc-rio.br/luaorb/pub/luaorb.pdf.

[35] TEINIKER, E.; MITTERDORFER, S.; JOHNSON, L. M.; KREINER, C.; KOVÁCS, Z. ; WEISS, R.. **A test-driven component development framework based on the CORBA component model**. Em: $27^{TH}$ INTERNATIONAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE (COMPSAC 2003): DESIGN AND ASSESSMENT OF TRUSTWORTHY SOFTWARE-BASED SYSTEMS, 3-6 NOVEMBER 2003, DALLAS, TX, USA, PROCEEDINGS, p. 400–405, Dalas, EUA, novembro 2003. IEEE Computer Society, IEEE Press. ISBN: 0-769-52020-0.

[36] RODRIGUEZ, N. R.; IERUSALIMSCHY, R. ; CERQUEIRA, R. F. G.. **Dynamic configuration with CORBA components**. Em: Cole, R.; Schlichting, R., editors, $4^{TH}$ INTERNATIONAL CONFERENCE ON CONFIGURABLE DISTRIBUTED SYSTEMS, ICCDS '98, p. 27–34, Anápolis, EUA, março 1998. IEEE Computer Society, IEEE Press. ISBN: 0-818-68451-8.

[37] KON, F.; CAMPBELL, R. H.. **Dependence management in component-based distributed systems**. IEEE Concurrency, 8(1):26–36, janeiro 2000.

[38] PILHOFER, F.. **The MICO CORBA component project**. Disponível em <http://www.fpx.de/MicoCCM/>. Acesso em: 30 de julho de 2004.

[39] TOSIC, V.; PAGUREK, B.; ESFANDIARI, B. ; PATEL, K.. **On various approaches to dynamic adaptation of distributed component compositions**. Relatório Técnico OCIECE-02-02, Ottawa-Carleton Institute for Electrical and Computer Engineering (OCIECE), Ottawa, Canadá, junho 2002. 10 p.

[40] BATISTA, T. V.; RODRIGUEZ, N. R.. **Dynamic reconfiguration of component-based applications**. Em: $5^{TH}$ INTERNATIONAL SYMPOSIUM ON SOFTWARE ENGINEERING FOR PARALLEL AND DISTRIBUTED SYSTEMS (PDSE 2000), p. 32–39, Limerick, Irlânda, junho 2000. IEEE Computer Society, IEEE Press. ISBN: 0-769-50634-8.

[41] RODRIGUEZ, N. R.; IERUSALIMSCHY, R.. **Dynamic reconfiguration of CORBA-based applications**. Em: Jan Pavelka, Gerard Tel, M. B., editor, SOFSEM'99: THEORY AND PRACTICE OF INFORMATICS: $26^{TH}$ CONFERENCE ON CURRENT TRENDS IN THEORY AND PRACTICE OF INFORMATICS, MILOVY, CZECH REPUBLIC, NOVEMBER 27–DECEMBER 4 1999. PROCEEDINGS, volume 1725 de **Lecture Notes in Computer Science**, p. 95–111, Milovy, República Tcheca, novembro 1999. Springer-Verlag Heidelberg. ISBN: 3-540-66694-X.

[42] Object Management Group, Needham, EUA. **CORBA Scripting Language Specification**, fevereiro 2003. document: formal/2003-02-01.

[43] MARVIE, R.; MERLE, P. ; GEIB, J.-M.. **Towards a dynamic CORBA component platform**. Em: INTERNATIONAL SYMPOSIUM ON DISTRIBUTED OBJECTS AND APPLICATIONS, p. 305–314, Antwerp, Bélgica, setembro 2000. IEEE Press.

[44] MERLE, P.; GRANSART, C. ; GEIB, J.-M.. **Using and implementing CORBA objects with CorbaScript**. Em: $2^{ND}$ FRANCE-JAPAN WORKSHOP ON OBJECT BASED PARALLEL AND DISTRIBUTED COMPUTING, OBPDC'97, Toulouse, França, outubro 1997. Hermes.

[45] MARTINS, M. C.; RODRIGUEZ, N. R. ; IERUSALIMSCHY, R.. **Dynamic extension of CORBA servers**. Em: Amestoy, P.; Berger, P.; Daydé, M.; Duff, I.; Frayssé, V.; Giraud, L. ; Ruiz, D., editors, EURO-PAR'99 - PARALLEL PROCESSING: $5^{TH}$ INTERNATIONAL EURO-PAR CONFERENCE, TOULOUSE, FRANCE, AUGUST 31–SEPTEMBER 3 1999. PROCEEDINGS, volume 1685 de **Lecture Notes in Computer Science**, p. 1369–1376, Toulouse, Franća, agosto 1999. ACM, Springer-Verlag Heidelberg. ISBN: 3-540-66443-2.

[46] PESCHANSKI, F.. **A versatile event-based communication model for generic distributed interactions**. Em: Wagner, R., editor, $22^{ND}$ INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS WORKSHOPS: 2–5 JULY 2002 VIENNA, AUSTRIA: PROCEEDINGS, p. 503–510, Viena, Áustria, julho 2002. IEEE Computer Society, IEEE Press. ISBN: 0-769-51588-6.

[47] TRUYEN, E.; VANHAUTE, B.; JØRGENSEN, B. N.; JOOSEN, W. ; VERBAETON, P.. **Dynamic and selective combination of extensions in component-based applications**. Em: SOFTWARE ENGINEERING (ICSE 2001): $23^{RD}$ INTERNATIONAL CONFERENCE, p. 233–242, Toronto, Canadá, maio 2001. IEEE Computer Society, IEEE Press. ISBN: 0-769-51050-7.

# Apêndice A

# Arquitetura CORBA

A arquitetura CORBA é uma especificação que define um ambiente para construção de aplicações integradas através de ambientes computacionais heterogêneos e distribuídos. Através dessa arquitetura é possível construir aplicações utilizando objetos implementados em plataformas distintas e escritos em linguagens de programação diferentes. Um objeto CORBA é uma implementação de um serviço que é acessado através de operações de uma interface bem definida, por meio de um mecanismo similar à chamada remota de procedimentos (*remote procedure call* - RPC). O conceito de objetos de CORBA define um modelo de componentes onde os conectores são representados pelas interfaces implementadas por cada objeto, que são utilizadas para acessar os seus serviços. Entretanto, esse modelo de componentes imposto pelos objetos CORBA não define mecanismos para representar explicitamente as dependências de um componente [41].

## A.1

## Arquitetura de Gerência de Objetos

A arquitetura CORBA é um componente de uma arquitetura maior chamada de Arquitetura de Gerência de Objetos (OMA - *Object Management Architecture*). A arquitetura OMA é formada por dois modelos: o modelo de objeto que define como objetos em diferentes plataformas podem ser descritos; e o de referência, que define como as interações entre esses objetos é feita. No modelo de objetos de OMA, objetos são entidades com identidade única e imutável, cujo serviços são acessíveis através de interfaces bem definidas. A figura A.1 ilustra a arquitetura OMA.

A arquitetura CORBA define o componente ORB - *Object Request Broker* da arquitetura OMA, que é responsável por tornar a comunicação entre objetos e clientes o mais simples possível. Os objetos que utilizam o ORB são agrupados em quatro categorias de acordo com suas interfaces.

**Serviços de Objetos** São objetos que fornecem serviços genéricos, que são utilizados por diferentes programas de objetos distribuídos, independente do domínio de aplicação. Esses serviços incluem serviços de descoberta, gerência do ciclo de vida, segurança, transações, notificação de eventos, entre outros.

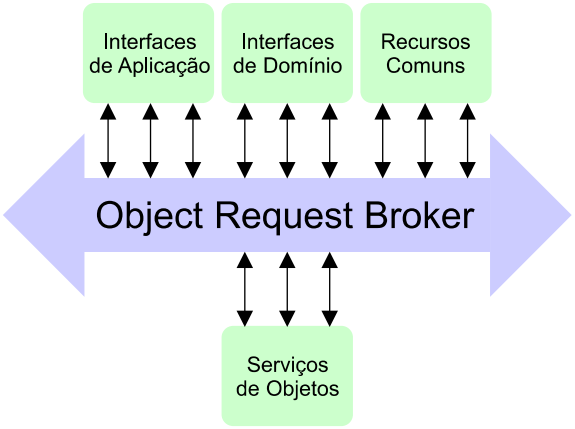


Figura A.1: Arquitetura OMA.

**Recursos Compartilhados** Assim como Serviços de Objetos, os Recursos Compartilhados também são independentes de domínio de aplicação, mas são destinados a serem utilizados por aplicações do usuário final. Um exemplo de tal recurso é o *Distributed Document Component Facility*, um recurso compartilhado de documentos compostos baseado no OpenDoc[1].

**Interfaces de Domínio** Essas interfaces possuem o mesmo propósito dos Serviços de Objetos e Recursos Compartilhados, mas são destinados a domínios de aplicação específicos. Há diversas interfaces de domínio nos campos das telecomunicações, medicina e finanças.

**Interfaces de Aplicação** Essas são as interfaces desenvolvidas especificamente para uma determinada aplicação. Essas interfaces não são cobertas pelas especificações da OMG.

Embora em uma única requisição, cada objeto funcione apenas como cliente ou como servidor, os objetos que utilizam o ORB se comunicam numa estrutura ponto a ponto, uma vez que cada objeto tem a capacidade de funcionar tanto como cliente como servidor. Assim, é possível desenvolver aplicações totalmente descentralizadas e inclusive utilizar outros modelos de comunicação, como por exemplo a comunicação baseada em eventos.

## A.2

## Estrutura do ORB

O ORB é responsável por entregar as chamadas feitas pelo cliente ao objeto servidor de forma transparente, facilitando a comunicação entre eles. O ORB é responsável por ocultar as seguintes características dos objetos:

**Localização** O cliente não sabe se o objeto está no mesmo processo ou se está em uma máquina diferente.

[1]OpenDoc é marca registrada da Apple Computer, Inc.

**Implementação** O cliente não sabe em que linguagem o objeto foi implementado ou em que plataforma está executando.

**Estado de execução** O cliente não precisa saber se o objeto está ativado e pronto para receber chamadas. O ORB automaticamente ativa o objeto, se necessário, antes de efetuar a chamada.

**Mecanismos de comunicação** O cliente não sabe que mecanismos de comunicação (*e.g.*, TCP/IP, compartilhamento da memória local, etc.) o ORB utiliza para enviar as chamadas e receber os resultados.

No ORB, os objetos possuem uma identificação única e imutável, que lhes é atribuída no momento de sua criação, chamada *referência de objeto*. Essas referências são opacas, ou seja, o seu conteúdo não é conhecido pelo cliente, apenas o ORB conhece sua estrutura. Os clientes podem obter referências de objeto de várias formas possíveis: como valor de retorno de uma operação, através de serviços de obtenção de referências (*i.e.*, Serviço de Nomes, Serviço de *Trading*), a partir de um arquivo contendo a referência de objeto em formato de texto, entre outras.

Como o ORB não fornece operações para que clientes criem objetos, todas as formas para obtenção de referências são feitas por intermédio de outros objetos. Por isso é necessário que o ORB forneça algum mecanismo para obtenção de uma referência de objeto inicial. Para tanto, o ORB fornece a operação resolve_initial_references, através da qual é possível, por exemplo, obter uma referência de um serviço de descoberta.

## A.3

## Linguagem de Definição de Interfaces

Para que os clientes possam utilizar os serviços de um objeto, é necessário que eles conheçam sua interface. Para isso, a arquitetura CORBA define uma linguagem de definição de interfaces chamada IDL - *Interface Definition Language*. Através dessa linguagem é possível definir interfaces compostas por operações, atributos, etc. de forma similar à declaração de classes abstratas de C++ e interfaces de Java.

Uma importante característica da linguagem IDL é ser uma linguagem de definição de interfaces com conceitos genéricos o suficiente para que possa ser facilmente mapeada para os conceitos da maioria das linguagens de programação. Isso permite que clientes em diferentes linguagens possam utilizar o mesmo objeto dada sua interface descrita em IDL. A linguagem IDL define tipos similares aos encontrados na maior parte das linguagens de programação, como long, double, boolean e inclusive tipos estruturados como struct, union, sequence, etc. Além disso, a linguagem IDL também permite a definição exceções e de operações que lançam exceções. Para evitar a colisão de nomes, a linguagem IDL define o conceito de módulos, que são espaços de nomes onde inter-

faces e novos tipos podem ser declarados. A linguagem IDL também permite que as interfaces possam herdar características de uma ou mais interfaces.

Para que um cliente possa manipular a interface de um objeto, é necessário que as interfaces definidas em IDL sejam mapeadas para os conceitos da linguagem de programação do cliente. Esse mapeamento também é necessário para permitir que os objetos que implementam uma interface sejam escritos numa determinada linguagem de programação, uma vez que esses objetos também precisam manipular a interface. A forma como esse mapeamento é feito é especificada pela arquitetura CORBA. Atualmente, já existe o mapeamento de IDL para a maioria das linguagens existentes incluindo C, C++, Java, COBOL, entre outras. O mapeamento para cada linguagem é automaticamente feito por compiladores de IDL, que convertem os tipos e interfaces IDL para os conceitos da linguagem de programação, gerando arquivos de código fonte que são compilados junto com a aplicação.

## A.4

## Stubs e Esqueletos



As chamadas através do ORB são feitas por intermédio de elementos que funcionam como representantes do cliente e do objeto em cada plataforma na qual uma chamada é feita, são eles o *stub* e o *esqueleto*. O *stub* representa o objeto servidor na plataforma do cliente, que recebe uma requisição e a transmite através do ORB fazendo a conversão das informações entre as plataformas. O esqueleto é o elemento que recebe uma requisição enviada pelo *stub*, a converte para a plataforma do servidor e entrega a requisição no lugar do cliente real, devolvendo posteriormente ao *stub* os resultados da requisição. Como a comunicação entre o *stub* e o esqueleto é transparente ao cliente e o objeto, a distribuição e a heterogeneidade das plataformas também se torna transparente.

O *stub* e o esqueleto são gerados pelo compilador de IDL, juntamente com o mapeamento dos tipos e interfaces IDL para uma determinada linguagem de programação. Dessa forma, esses dois elementos são gerados com todas informações sobre as interfaces do objeto e os tipos de dados trocados através das interfaces, que são utilizadas na transmissão das requisições entre as plataformas. Como essa informação sobre os tipos e as interfaces é embutida no código do *stub* e do esqueleto, essas chamadas são ditas estáticas. Por essa razão, sempre que as interfaces ou tipos entre os objetos se modificam, é necessário, gerar novamente a aplicação com novos *stubs* e esqueletos compilados a partir das novas interfaces e tipos. A figura A.2 ilustra o funcionamento dos *stubs* e o esqueletos da arquitetura CORBA.

A arquitetura CORBA também define um mecanismo que permite fazer chamadas com base em informações de interfaces e tipos obtidas dinamicamente através do serviço de objetos denominado *Repositório de Interfaces*. As chamadas feitas a partir desse mecanismo são ditas dinâmicas e serão apresentadas com maiores detalhes na seção A.6

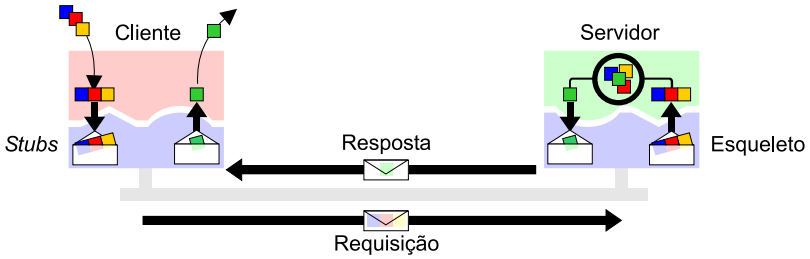


Figura A.2: Funcionamento dos *stubs* e esqueletos da arquitetura CORBA.

## A.5
## Repositório de Interfaces

Para converter os valores de um chamada CORBA, é necessário ter informações sobre os tipos e interfaces IDL que formam o sistema de tipos de CORBA. Essa informação pode ser estaticamente definida, como no caso dos *stubs* e esqueletos gerados pelo compilador IDL, ou pode ser dinamicamente obtida a partir do Repositório de Interfaces. O Repositório de Interfaces é um serviço de objetos da arquitetura CORBA e é acessado como um objeto CORBA qualquer, ou seja, as informações contidas no repositório são acessadas através de um conjunto de interfaces descritas em IDL.

Todas as informações descritas através da linguagem IDL, podem ser obtidas do Repositório de Interfaces, através de uma estrutura hierárquica. Por exemplo, a partir de uma determinada interface é possível obter o módulo a que pertence, assim como iterar sobre todos os seus membros, como operações, atributos e inclusive outras interfaces. Além disso, as interfaces do Repositório de Interfaces também permitem alterar a as informações contidas no repositório, funcionando como um mecanismo de reflexão computacional, como é apresentado no capítulo 3.

## A.6
## Chamadas Dinâmicas

As chamadas dinâmicas de CORBA são feitas através de dois mecanismos da arquitetura: a interface de invocação dinâmica e a interface de esqueleto dinâmico.

A interface de invocação dinâmica (*Dynamic Invokation Interface* — DII) permite criar requisições através da operação create_request presente em todos objetos CORBA, onde é possível definir o nome da operação, assim como os valores dos seus parâmetros e então efetuar a invocação da operação. O número de parâmetros de uma chamada e seus respectivos tipos podem ser obtidos através do Repositório de Interfaces. Dependendo do modo de invocação utilizado, a chamada pode ficar bloqueada até que o processamento no servidor seja completado e os resultados devolvidos (chamada síncrona), ou a chamada

pode ser encerrada assim que a requisição seja enviada e posteriormente pode-se verificar se a operação foi concluída (chamada síncrona postergada).

Assim como a interface de invocação dinâmica permite criar requisições de forma dinâmica, a interface de esqueleto dinâmico (*Dynamic Skeleton Interface* — DSI) permite tratar requisições de forma dinâmica. Através desse mecanismo, é possível receber quaisquer requisições para um objeto ou grupo de objetos, podendo acessar o identificador do objeto, o nome da operação, assim como os valores dos parâmetros. Tal recurso pode ser utilizado em diversas aplicações, em especial na construção de objetos cuja implementação possa mudar, como é visto no capítulo 4.