Referências Bibliográficas

[01] CARR, H.. **PEPt - A Minimal RPC Architecture**. In: OTM WORKSHOPS, p. 109–122, 2003. 1, 2.2.1, 2.2.1

[02] MICROSOFT CORPORATION. **.NET Framework Home**. http://msdn.microsoft.com/netframework/. 1, 2.2.2

[03] BLAIR, G. S.; COULSON, G.; ROBIN, P. ; PAPATHOMAS, M.. **An architecture for next generation middleware**. In: PROCEEDINGS OF THE IFIP INTERNATIONAL CONFERENCE ON DISTRIBUTED SYSTEMS PLATFORMS AND OPEN DISTRIBUTED PROCESSING, London, 1998. Springer-Verlag. 1, 2.1.2

[04] HAUCK, F.; BECKER, U.; GEIER, M.; MEIER, E.; RASTOFER, U. ; STECKMEIER, M.. **AspectIX: An Aspect-Oriented and CORBA-Compliant ORB Architecture**, 1998. 1

[05] IERUSALIMSCHY, R.. **Programming in Lua, Second Edition**. Lua.Org, 2006. 1, 3

[06] GROUP OF DISTRIBUTED SYSTEMS - PUC-RIO. **ORB in Lua - OiL**. http://oil.luaforge.net/, 2005. 1

[07] DE MOURA, A. L.; IERUSALIMSCHY, R.. **Revisiting coroutines**. Technical Report 15/04, PUC-Rio, Rio de Janeiro, RJ, June 2004. 1, 4.1.4

[08] CAMPBELL, A. T.; COULSON, G. ; KOUNAVIS, M. E.. **Managing Complexity: Middleware Explained**. IT Professional, 1(5):22–28, 1999. 2

[09] EMMERICH, W.. **Software engineering and middleware: a roadmap**. In: ICSE - FUTURE OF SE TRACK, p. 117–129, 2000. 2

[10] HUDDERS, E. S.. **CICS: a guide to internal structure**. Wiley-QED Publishing, Somerset, NJ, USA, 1994. 2

[11] HALL, C. L.. **Building client/server applications using TUXEDO**. John Wiley & Sons, Inc., New York, NY, USA, 1996. 2

[12] GILMAN, L.; SCHREIBER, R.. **Distributed Computing with IBM MQSeries**. John Wiley & Sons, Inc., New York, NY, USA, 1996. 2

[13] HAPNER, M.; BURRIDGE, R. ; SHARMA, R.. **Java message service specification**. Technical report, Sun Microsystems, November 1999. 2

[14] BROWN, N.; KINDEL, C.. **Distributed Component Object Model Protocol–DCOM 1.0**. Technical report, Microsoft Corporation, 1996. 2

[15] OMG: OBJECT MANAGEMENT GROUP. **Common Object Request Broker Architecture Specification**. http://www.omg.org/corba. 2

[16] VINOSKI, S.. **CORBA: integrating diverse applications within distributed heterogeneous environments**. IEEE Communications Magazine, 14(2), 1997. 2

[17] WOLLRATH, A.; RIGGS, R. ; WALDO, J.. **A distributed object model for the Java System**. In: 2ND CONFERENCE ON OBJECT-ORIENTED TECHNOLOGIES & SYSTEMS (COOTS), p. 219–232. USENIX Association, 1996. 2

[18] NARASIMHAN, P.; MOSER, L. E. ; M. MELLIAR-SMITH, P.. **Using Interceptors to Enhance CORBA**. Computer, 32(7):62–68, 1999. 2.1

[19] SADJADI, S. M.; MCKINLEY, P. K.. **A Survey of Adaptive Middleware**. Technical Report MSU-CSE-03-35, Computer Science and Engineering, Michigan State University, East Lansing, Michigan, December 2003. 2.1

[20] MCKINLEY, P. K.; SADJADI, S. M.; KASTEN, E. P. ; CHENG, B. H. C.. **A Taxonomy of Compositional Adaptation**. Technical Report MSU-CSE-04-17, Department of Computer Science and Engineering, Michigan State University East Lansing, Michigan, 2004. 2.1, 2.2

[21] SCHMIDT, D. C.. **Middleware for real-time and embedded systems**. Commun. ACM, 45(6):43–48, 2002. 2.1.1, 2.1

[22] KICZALES, G.; RIVIERES, J. D.. **The Art of the Metaobject Protocol**. MIT Press, Cambridge, MA, USA, 1991. 2.1.2

[23] WATANABE, T.; YONEZAWA, A.. **Reflection in an object-oriented concurrent language**. In: OOPSLA '88: CONFERENCE PROCEEDINGS ON OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES AND APPLICATIONS, p. 306–315, New York, NY, USA, 1988. ACM Press. 2.1.2

[24] AGHA, G.. **The structure and semantics of actor languages**. In: de Bakker, J. W.; de Roever, W. P. ; Rozenberg, G., editors, FOUNDATIONS OF OBJECT-ORIENTED LANGUAGES, p. 1–59. Springer, Berlin, Heidelberg, 1991. 2.1.2

[25] YOKOTE, Y.. **The apertos reflective operating system: the concept and its implementation**. In: OOPSLA '92: CONFERENCE PROCEEDINGS ON OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES, AND APPLICATIONS, p. 414–434, New York, NY, USA, 1992. ACM Press. 2.1.2

[26] KASTEN, E. P.; MCKINLEY, P. K.; SADJADI, S. M. ; STIREWALT, K.. **Separating introspection and intercession to support metamorphic distributed systems**. In: ICDCSW '02: PROCEEDINGS OF THE 22ND INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, p. 465–472, Washington, DC, USA, 2002. IEEE Computer Society. 2.1.2

[27] SZYPERSKI, C.. **Component Software: Beyond Object-Oriented Programming**. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002. 2.1.2

[28] SUN MICROSYTEMS. **EJB: Enterprise Java Beans**. http://java.sun.com/products/ejb. 2.1.2

[29] WANG, N.; SCHMIDT, D. C. ; O'RYAN, C.. **Overview of the CORBA component model**. p. 557–571, 2001. 2.1.2

[30] KICZALES, G.; HILSDALE, E.. **Aspect-oriented programming**. In: ESEC/FSE-9: PROCEEDINGS OF THE 8TH EUROPEAN SOFTWARE ENGINEERING CONFERENCE HELD JOINTLY WITH 9TH ACM SIGSOFT INTERNATIONAL SYMPOSIUM ON FOUNDATIONS OF SOFTWARE ENGINEERING, p. 313, New York, NY, USA, 2001. ACM Press. 2.1.2

[31] KICZALES, G.; HILSDALE, E.; HUGUNIN, J.; KERSTEN, M.; PALM, J. ; GRISWOLD, W. G.. **An Overview of AspectJ**. In: ECOOP '01: PROCEEDINGS OF THE 15TH EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, p. 327–353, London, UK, 2001. Springer-Verlag. 2.1.2

[32] POPOVICI, A.; GROSS, T. ; ALONSO, G.. **Dynamic homogenous aop with prose**, 2001. 2.1.3

[33] DE OLIVEIRA VALENTE, M. T.; TIRELO, F.; LEAO, D. C. ; SILVA, R. P.. **An aspect-oriented communication middleware system.** In: OTM CONFERENCES (2), p. 1115–1132, 2005. 2.1.3, 2.2.4

[34] WELCH, I.; STROUD, R. J.. **Kava - A Reflective Java Based on Bytecode Rewriting**. In: PROCEEDINGS OF THE 1ST OOPSLA WORKSHOP ON REFLECTION AND SOFTWARE ENGINEERING, p. 155–167, London, UK, 2000. Springer-Verlag. 2.1.3

[35] CARR, H.. **ContactInfo - Client-side Encoding, Protocol and Transport Extensibility for Remoting Systems**. In: ICSOC '04: PROCEEDINGS OF THE 2ND INTERNATIONAL CONFERENCE ON SERVICE ORIENTED COMPUTING, p. 329–334, New York, NY, USA, 2004. ACM Press. 2.2.1, 2.5

[36] CARR, H.. **Acceptor - Server-side Encoding, Protocol and Pransport Extensibility for Remoting Systems**. In: CIC '04: PROCEEDINGS OF THE FIFTH INTERNATION CONFERENCE ON COMMUNICATIONS IN COMPUTING, 2004. 2.2.1, 2.2.1

[37] OBERMEYER, P.; HAWKINS, J.. **Microsoft .NET Remoting: A Technical Overview**. http://msdn.microsoft.com/library, July 2001. 2.2.2

[38] MCLEAN, S.; WILLIAMS, K. ; NAFTEL, J.. **Microsoft .Net Remoting**. Microsoft Press, Redmond, WA, USA, 2002. 2.2.2

[39] **Remoting.Corba**. http://remoting-corba.sourceforge.net/. 2.2.2

[40] MICROSOFT CORPORATION. **COM: Component Object Model Technologies**. http://www.microsoft.com/com. 2.2.3

[41] YI-MIN, W.; WOEI-JYH, L.. **COMERA: COM extensible remoting architecture**. In: PROCEEDINGS OF THE 4TH USENIX CONFERENCE ON OBJECT-ORIENTED TECHNOLOGIES AND SYSTEMS (COOTS). USENIX, 1998. 2.7, 2.8, 2.2.3

[42] THE OPEN GROUP. **DCE 1.1: Remote Procedure Call Specification**. http://www.rdg.opengroup.org/public/pubs/catalog/c706.htm. 2.2.3

[43] PEREIRA, F. M.; VALENTE, M. T.; BIGONHA, R. ; BIGONHA, M.. **Chamada remota de métodos na plataforma J2ME/CLDC**. In: V WORKSHOP DE COMUNICAÇÃO SEM FIO E COMPUTAÇÃO MÓVEL, 2003. 2.2.4

[44] GAMMA, E.; HELM, R.; JOHNSON, R. ; VLISSIDES, J.. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison Wesley, Reading, Massachusetts, 1994. 2.2.4

[45] BRUNETON, E.; COUPAYE, T. ; STEFANI, J.. **The Fractal Project**. http://fractal.objectweb.org. 2.3.1, 2.9

[46] BRUNETON, E.. **The Fractal Project – Julia Tutorial**. http://fractal.objectweb.org/tutorials/julia/index.html. 2.3.1

[47] SEINTURIER, L.; PESSEMIER, N. ; COUPAYE, T.. **The Fractal Project – AOKell 2.0 Documentation**. http://fractal.objectweb.org/tutorials/aokell/index.html. 2.3.1

[48] COULSON, G.; BLAIR, G. S.; CLARKE, M. ; PARLAVANTZAS, N.. **The design of a configurable and reconfigurable middleware platform**. Distrib. Comput., 15(2):109–126, 2002. 2.3.2, 2.10, 2.3.2, 2.11

[49] HAUCK, F. J.; MEIER, E.; BECKER, U.; GEIER, M.; RASTOFER, U. ; STECKERMEIER, M.. **A middleware architecture for scalable, QoS-aware, and self-organizing global services**. In: PROCEEDINGS OF THE 3RD INTERNATIONAL IFIP/GI WORKING CONFERENCE, USM (USM), volumen 1890, p. 214–229, Berlin, Heidelberg, New York, Tokyo, 2000. Springer-Verlag. 2.3.3, 2.12

[50] HAUCK, F. J.; BECKER, U.; GEIER, M.; MEIER, E.; RASTOFER, U. ; STECKERMEIER, M.. **AspectIX: a quality-aware, object-based middleware architecture**. In: PROC. OF THE 3RD IFIP INT. CONF. ON DISTRIB. APPL. AND INTEROPERABLE SYS. Kluver, 2001. 2.3.3

[51] VAN STEEN, M.; HOMBURG, P. ; TANENBAUM, A. S.. **The Architectural Design of Globe: A Wide-Area Distributed System**. Technical Report IR-422, Netherlands, 1997. 2.3.3

[52] NEHAB, D.. **Network support for the Lua language**. http://luaforge.net/projects/luasocket/, 2004. 3, A

[53] MAIA, R.; CERQUEIRA, R. ; KON, F.. **A Middleware for Experimentation on Dynamic Adaptation**. In: ACM/IFIP/USENIX 3RD INTERNATIONAL WORKSHOP ON ADAPTIVE AND REFLECTIVE MIDDLEWARE, Grenoble, France, November 2005. 3, 4

[54] COSTA, A. T.; ENDLER, M. ; CERQUEIRA, R.. **Evaluation of Three Approaches for CORBA Firewall/NAT Traversal**. In: OTM CONFERENCES (2), p. 923–940, 2005. 3, 6.1

[55] GOLDCHLEGER, A.; KON, F.; GOLDMAN, A.; FINGER, M. ; BEZERRA, G. C.. **InteGrade object-oriented Grid middleware leveraging the idle computing power of desktop machines: Research Articles**. Concurr. Comput. : Pract. Exper., 16(5):449–459, 2004. 3

[56] DE MELLO, R. X.. **Um modelo de escalonamento colaborativo de eventos baseado em corrotinas**. Dissertação de Mestrado, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Rio de Janeiro, Brasil, Setembro de 2005. PUC-Rio. 3, 4.1.4

[57] MAIA, R.. **Um Framework para Adaptação Dinâmica de Sistemas Baseados em Componentes Distribuídos**. Dissertação de Mestrado, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Rio de Janeiro, Brasil, Março de 2004. PUC-Rio. 3

[58] XENOS, M.; STADVRINOUDIS, D.; ZIKOULI, K. ; CHRISTODOULAKIS, D.. **Object-oriented metrics - a survey**. In Proceedings of the FESMA 2000., 2000. 6.1

[59] SANT'ANNA, C. N.. **Manutenibilidade e Reusabilidade de Software Orientado a Aspectos: Um Framework de Avaliação**. Dissertação de Mestrado, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Rio de Janeiro, Brasil, Março de 2004. PUC-Rio. 6.1

[60] HENNING, M.. **A New Approach to Object-Oriented Middleware**. IEEE Internet Computing, 8(1):66–75, 2004. 6.1

A
API dos componentes

ProtocolType

Receptáculos

channels

Ligado à faceta *factory* de `ChannelFactoryType`.

codec

Ligado à faceta *codec* de `CodecType`.

InvokeProtocolType

Subtipo de `ProtocolType`.

Facetas

invoker

Função: *sendrequest*
Descrição: Faz uma chamada remota.
Parâmetros:

reference	Tabela contendo a referência para o objeto remoto.
operation	Nome da operação a ser invocada no objeto remoto.
parameters	Parâmetros a serem enviados à chamada remota.

Retornos:

reply	Objeto composto por dois elementos: *success*, um boleano que diz se a operação teve sucesso ou não; e *result*, função que deve ser chamada para obter os valores de retorno da chamada remota.

Receptáculos

tasks

Ligado à faceta *threads* de `SchedulerType`.

ListenProtocolType

Subtipo de `ProtocolType`.

Facetas

listener

Função: *getchannel*

Descrição: Obtém um canal de comunicação que aguarda por uma requisição.

Parâmetros:

args	Tabela contendo as informações necessárias para a criação de um canal. No caso de um canal TCP/IP, a tabela contém dois campos, o *host* e *port*, com o nome ou IP e a porta que devem ser usadas para o *socket* começar a escutar. O *host* pode ser omitido, sendo que nesse caso ele será substituído por "*".

Retornos:

portConn	Objeto que contém o *socket* criado e que serve de fachada para as operações de escrita e leitura desse *socket*, realizando tarefas adicionais caso sejam necessárias. Por exemplo, no GIOP, ao fechar o *socket* este objeto tenta enviar uma mensagem de "Close Connection" do protocolo ao objeto remoto antes de fechar o *socket* local.
except	Caso a operação de criação do canal não funcione, o primeiro retorno será nulo e este retorno terá uma exceção com a explicação do erro.

Função: *getrequest*
Descrição: Obtém um objeto de requisição de um canal de comunicação.
Parâmetros:

channel	Um portConn que tenha sido criado através de *getchannel*.

Retornos:

request	Objeto composto por dois elementos: *success*, um boleano que diz se a operação teve sucesso ou não; e *result*, função que deve ser chamada para obter os valores de retorno da chamada remota. Se houver algum erro, este objeto será nulo.
except	Exceção, caso haja erro na obtenção de uma requisição.

TypedInvokeProtocolType

Subtipo de `InvokeProtocolType`.

Receptáculos

interfaces

Ligado à faceta *registry* de `TypeManagerType`.

TypedListenProtocolType

Subtipo de `ListenProtocolType`.

Receptáculos

objects

Ligado à faceta *registry* de `TypeManagerType`.

CodecType

Facetas

codec

Função: *newEncoder*
Descrição: Obtém um objeto codificador.
Parâmetros:

...	Parâmetros adicionais para a criação de um objeto codificador específico do tipo de codificação desejado. Por exemplo, no caso de CORBA, é possível passar um parâmetro *order*, que diz a ordem de codificação (*little endian* ou *big endian*).

Retornos:

encoder	Objeto usado para codificar dados. Esse objeto possui funções para codificação dos tipos usados pelo protocolo, como por exemplo *boolean* e *string*.

Função: *newDecoder*
Descrição: Obtém um objeto decodificador.
Parâmetros:

octets	*Bytes* que devem ser decodificados por este objeto.
...	Parâmetros adicionais para a criação de um objeto decodificador específico do tipo de codificação desejado. Por exemplo, no caso de CORBA, é possível passar um parâmetro *order*, que diz a ordem de decodificação (*little endian* ou *big endian*).

Retornos:

decoder	Objeto usado para decodificar os *bytes* passados como parâmetro. Esse objeto possui funções para decodificação dos tipos usados pelo protocolo, como por exemplo *boolean* e *string*.

ChannelFactoryType

Facetas

factory

Função: *create*
Descrição: Cria um canal de transporte.
Parâmetros:

configs	Configurações para a criação de um canal de transporte, dependendo do tipo de transporte desejado. Para conexões TCP/IP, são passados o *host* e a porta desejados.

Retornos:

channel	Canal criado com as configurações passadas como parâmetro. Este tipo possui duas implementações, o *ActiveChannelFactory* e o *PassiveChannelFactory*. Com isso, o canal devolvido pode ser um canal ativo, usado para fazer requisições ou um canal passivo, usado para escutar por requisições, dependendo do tipo desejado de utilização.

Receptáculos

luasocket

Ligado à faceta *socket* de `SchedulerType`.

ReferenceResolverType

Facetas

resolver

Função: *resolve*
Descrição: Decodifica uma referência para um objeto remoto
Parâmetros:

ref	Referência codificada para o objeto remoto.

Retornos:

decRef	Tabela com as informações necessárias para contactar o objeto remoto.

Função: *referto*
Descrição: Codifica uma tabela de referência de um objeto remoto para uma *string*.
Parâmetros:

servant	*Servant* cuja referência vai ser codificada.
info	Tabela com outras informações relacionadas ao *servant* que serão usadas pelo codificador de referências, tais como *host* e porta desejadas.

Retornos:

strRef	*String* com a referência para este objeto remoto passado como parâmetro

Receptáculos

codec

Ligado à faceta *codec* de `CodecType`.

ProxyFactoryType

Facetas

proxies

Função: *create*
Descrição: Cria um *proxy* para um objeto remoto.
Parâmetros:

reference	Referência decodificada para o objeto remoto para o qual vai ser criado o *proxy*.
protocol	Protocolo a ser utilizado nessa comunicação entre os objetos.
intfName	Nome da interface no repositório de interfaces, se houver (opcional).

Retornos:

object	*Proxy* criado. Ao serem feitas chamadas sobre esse objeto, ele as trasforma em chamadas remotas.

TypedProxyFactoryType

Subtipo de `ProxyFactoryType`

Receptáculos

interfaces

Ligado à faceta *registry* de `TypeManagerType`.

ClientBrokerType

Facetas

proxies

Função: *newProxy*

Descrição: Cria um *proxy* para um objeto remoto, usada pelo cliente do OiL.

Parâmetros:

textref	Referência codificada para o objeto remoto para o qual vai ser criado o *proxy*. Essa referência deve ser decodificada pelo decodificador de referências associado a este componente.
intfName	Nome da interface no repositório de interfaces, se houver (opcional).

Retornos:

object	*Proxy* criado.

Receptáculos

reference

Ligado à faceta *resolver* de *ReferenceResolverType*.

protocol

Ligado à faceta *invoker* de `InvokeProtocolType`.

factory

Ligado à faceta *proxies* de `ProxyFactoryType`.

AcceptorType

Facetas

manager

Função: *init*
Descrição: Inicializa um *servant*.
Parâmetros:

config	Configurações iniciais para o *servant*. Algumas das informações nessa tabela são o *host* e a porta associados a ele, dependendo do tipo de protocolo e transporte em que esses *servants* serão criados.

Retornos: Nenhum.
Função: *getinfo*
Descrição: Obtém informações de um *servant*.
Parâmetros: Nenhum.
Retornos:

info	Tabela com informações desse *servant*, para serem utilizadas pelo codificador de referências.

Função: *accept*
Descrição: Faz com que o *servant* aguarde por uma requisição.
Parâmetros: Nenhum.
Retornos:

result	Resultado da chamada remota feita a esse *servant*.
errmsg	Mensagem de erro no processamento, se houver.

Função: *acceptall*
Descrição: Faz com que o *servant* aguarde por inúmeras requisições.
Parâmetros: Nenhum.
Retornos:

result	Resultado da última chamada remota feita a esse *servant*.
errmsg	Mensagem de erro no processamento, se houver.

Receptáculos

listener

Ligado à faceta *listener* de `ListenProtocolType`.

dispatcher

Ligado à faceta *listener* de `DispatcherType`.

tasks

Ligado à faceta *threads* de `SchedulerType`.

DispatcherType

Facetas

registry

Função: *register*
Descrição: Registra um *servant* nesse *dispatcher*.
Parâmetros:

key	Identificador com o qual o *servant* vai ficar registrado.
object	O *servant* em si.
intfName	Nome da interface do *servant* no repositório de interfaces, se houver (opcional).

Retornos:

locObject	O objeto que foi registrado nesse *dispatcher*.

Função: *getobject*
Descrição: Obtém um *servant* registrado nesse *dispatcher* a partir do seu identificador.
Parâmetros:

key	Identificador do *servant*.

Retornos:

locObject	O objeto com o identificador passado como parâmetro, se ele existir.

Função: *deactivate*
Descrição: Apaga do registro deste *dispatcher* o *servant* com um dado identificador.
Parâmetros:

key	Identificador do *servant*.

Retornos: Nenhum.

dispatcher

Função: *handle*

Descrição: Recebe uma tabela com as informações da requisição para um objeto e, com isso, executa essa requisição e envia seu resultado.

Parâmetros:

request	Tabela com informações da requisição, como o identificador do objeto para o qual a requisição é feita, assim como o nome da requisição e os parâmetros que serão passados a ela.

Retornos:

success	Verdadeiro se a execução foi feita com sucesso e falso caso contrário.
result	Se há um resultado, uma tabela com os valores de retorno da operação. Caso haja um erro na execução, uma exceção é colocada neste valor de retorno.

Receptáculos

tasks

Ligado à faceta *threads* de `SchedulerType`.

TypedDispatcherType

Subtipo de `DispatcherType`.

Receptáculos

objects

Ligado à faceta *registry* de `TypeManagerType`.

ServerBrokerType

Facetas

registry

Função: *register*
Descrição: Registra a implementação de um objeto no servidor.
Parâmetros:

object	Implementação do objeto.
intfName	Nome da interface para a qual esse objeto é a implementação.
id	Identificador para o *servant* (opcional). Caso não seja colocado esse identificador, o servidor deve criar um próprio, que estará disponível para uso dentro do *servant*, retornado por essa função.

Retornos:

servant	*Servant* criado a partir dessa implementação.

Função: *unregister*
Descrição: Retira do registro o *servant* especificado.
Parâmetros:

servant	Referência para o *servant*.

Retornos:

result	Verdadeiro se o *servant* foi retirado do registro com sucesso ou falso caso contrário.

Função: *tostring*
Descrição: Obtém uma *string* com a referência codificada para o *servant*.
Parâmetros:

servant	Referência para o *servant*.
portName	Porta na qual o *servant* está registrado. Essa porta é relacionada com o protocolo, e por consequência ao tipo de codificação de referências.

Retornos:

textref	Referência codificada para o *servant*, usando o tipo de referência especificada na porta passada como parâmetro.

control

Função: *run*

Descrição: Executa o servidor, já configurado. Nesse momento o servidor começa a escutar por requisições nas portas criadas anteriormente e as executa na medida em que as recebe.

Parâmetros: Nenhum.

Retornos: Nenhum.

Função: *step*

Descrição: Espera por uma requisição, executa-a e retorna o controle para o código que chamou esta função. Serve para se ter um controle externo sobre o fluxo de execução deste servidor.

Parâmetros: Nenhum.

Retornos: Nenhum.

Função: *pending*

Descrição: Obtém a informação de se existe ou não uma requisição à espera de ser executada.

Parâmetros: Nenhum.

Retornos:

reqPend	Verdadeiro caso haja uma requisição à espera e falso caso contrário.

Receptáculos

ports

Receptáculo múltiplo ligado a uma ou mais facetas *listener* de `AcceptorType`.

objectmap

Ligado à faceta *registry* de `DispatcherType`.

objectmap

Receptáculo múltiplo ligado a uma ou mais facetas *resolver* de `ReferenceResolverType`.

TypeManagerType

Facetas

registry

Função: *update*
Descrição: Registra a interface de um objeto no repositório de interfaces.
Parâmetros:

typeid	Nome da interface.
def	Definição de quais operações e qual o número de parâmetros, bem como os tipos dos parâmetros existem na interface desse objeto.

Retornos: Nenhum.
lookup
Descrição: Obtém a interface de um objeto.
Parâmetros:

typeid	Nome da interface.

Retornos:

def	A definição da interface do objeto cujo nome foi dado como parâmetro.

SchedulerType

Facetas

threads

Função: *register*
Descrição: Registra a uma *thread* no escalonador.
Parâmetros:

routine	*Thread* a ser registrada no escalonador. Essa *thread* já deve ter sido inicializada com `coroutine.create`

Retornos:

result	Verdadeiro se a *thread* foi registrada com sucesso e falso caso contrário.

lookup

Função: *start*

Descrição: Cria implicitamente uma *thread* a partir de uma função e dos seus parâmetros.

Parâmetros:

func	Função a ser executada pela *thread*.
...	Argumentos a serem passados para a *thread*.

Retornos:

...	Os valores de retorno da função que foi executada pela *thread*.

Função: *remove*

Descrição: Remove uma *thread* do escalonador.

Parâmetros:

routine	*Thread* a ser removida do escalonador.

Retornos:

result	Verdadeiro se a *thread* foi registrada com sucesso e falso caso contrário.

Função: *resume*

Descrição: Continua a execução de uma *thread*.

Parâmetros:

routine	*Thread* cuja execução continuará.
...	Parâmetros enviados à *thread*.

Retornos:

...	Os valores passados para a *thread* que executou a função *suspend*.

Função: *suspend*

Descrição: Suspende a execução da *thread* atual, passando o controle para outra *thread* que estava suspensa no escalonador.

Parâmetros:

time	Tempo máximo em que essa *thread* deve ficar suspensa.

Retornos:

...	Os valores passados à *thread* que executou um *resume*.

socket

Exporta as funções da API da biblioteca LuaSocket(52).

B
Definição dos tipos dos componentes

```
1  local component = require "loop.component.base"
2  local port      = require "loop.component.base"
3
4  module "oil.arch.comm"
5
6  ProtocolType = component.Type{
7    channels = port.Receptacle,
8      -- channel create(<configs>)
9    codec = port.Receptacle,
10     -- encoder newencoder()
11     -- decoder newdecoder()
12 }
13
14 InvokeProtocolType = component.Type({
15   invoker = port.Facet,
16     -- [reply] sendrequest(reference, operation, <parameters>)
17     --  reply = {
18     --    success, <results or error> :result(),
19     --    boolean :probe()
20     --  }
21   tasks = port.Receptacle,
22 }, ProtocolType)
23
24 ListenProtocolType = component.Type({
25   listener = port.Facet,
26     -- channel getchannel(<configs>)
27     -- request getrequest(channel)
28     --  request = {
29     --    objectid = "Object Identifier",
30     --    operation = "operation name",
31     --    paramcount = <number of parameters>,
32     --    [1] = <first parameter>,
33     --    [2] = <second parameter>,
34     --    ...
35     --
36     --    :reply(success, <results or error>)
37     --  }
38     --
39 }, ProtocolType)
40
41 TypedInvokeProtocolType = component.Type({
42   interfaces = port.Receptacle,
43     -- interface lookup(interfaceid)
44 }, InvokeProtocolType)
45
46 TypedListenProtocolType = component.Type({
47   interfaces = port.Receptacle,
48     -- interface lookup(interfaceid)
```

```
}, ListenProtocolType)

CodecType = component.Type{
  codec = port.Facet,
    -- operations :
    --  newEncoder()
    --  newDecoder()
}

ChannelFactoryType = component.Type{
  factory = port.Facet,
    -- channel create(<configs>)
  luasocket = port.Receptacle,
}

SchedulerType = component.Type{
  threads = port.Facet,
  socket = port.Facet,
}

--- Reference
ReferenceResolverType = component.Type{
  resolver = port.Facet,
  codec = port.Receptacle,
}

--- Invocation
ProxyFactoryType = component.Type{
  proxies = port.Facet,
    -- proxy create(reference, protocol)
}

TypedProxyFactoryType = component.Type({
  interfaces = port.Receptacle,
    -- interface lookup(interfaceid)
    -- class getclass(interface)
}, ProxyFactoryType)

ClientBrokerType = component.Type{
  proxies = port.Facet,
    -- proxy create(textref, interfaceName)
  reference = port.Receptacle,
    -- reference resolve(textref)
  protocol = port.Receptacle,
    -- [reply] sendrequest(reference, operation, <parameters>)
  factory = port.Receptacle,
    -- proxy create(reference, protocol, interfaceName)
}

--- Servant
AcceptorType = component.Type{
  manager = port.Facet,
    -- init(<configs>)
    -- info getinfo()
    -- acceptall()
  listener = port.Receptacle,
    -- channel getchannel(<configs>)
    -- request getrequest(channel)
  dispatcher = port.Receptacle,
```

```
108     -- handle(request)
109   tasks = port.Receptacle,
110     -- start(function, ...)
111 }
112
113 DispatcherType = component.Type{
114   registry = port.Facet,
115     -- register(id, object)
116     -- object getobject(id)
117     -- object deactivate(id)
118   dispatcher = port.Facet,
119     -- handle(request)
120   tasks = port.Receptacle,
121     -- start(function, ...)
122 }
123
124 TypedDispatcherType = component.Type({
125   interfaces = port.Receptacle,
126     -- interface lookup(interfaceid)
127 }, DispatcherType)
128
129 ServerBrokerType = component.Type{
130   registry = port.Facet,
131     -- servant register(object, [id])
132     -- object unregister(servant)
133     -- textref tostring(servant)
134   control = port.Facet,
135     -- run()
136     -- step()
137     -- boolean pending()
138   ports = port.HashReceptacle,
139     -- acceptall()
140     -- accept()
141     -- boolean pending()
142   objectmap = port.Receptacle,
143     -- register(id, object)
144     -- object unregister(id)
145   reference = port.HashReceptacle,
146     -- textref referto(reference)
147 }
148
149 --- Interface
150 TypeManagerType = component.Type{
151   registry = port.Facet,
152     -- update(typeid, def)
153     -- def lookup(typeid)
154 }
```

Listagem B.1: Definição dos tipos de componentes da arquitetura do OiL

C
Arquivos de configuração dos exemplos

Apresentamos aqui os arquivos de configuração usados nos exemplos, mostrando os componentes carregados e as ligações entre eles.

Configuração simples usando CORBA

```
module "oil.configs.CORBASimple"
-----------------------------------------------------
local arch = require "oil.arch.comm"

local corba_codec          = require "oil.corba.Codec"
local corba_protocol       = require "oil.corba.Protocol"
local corba_reference      = require "oil.corba.reference"

local proxy              = require "oil.corba.proxy"
local client_broker      = require "oil.ClientBroker"
local server_broker      = require "oil.ServerBroker"
local channel_factory    = require "oil.ChannelFactorySelect"
local dispatcher         = require "oil.corba.SimpleDispatcher"
local manager            = require "oil.ir"
local access_point       = require "oil.SimpleAcceptor"

-----------------------------------------------------
local Factory_Codec              = arch.CodecType{ corba_codec }
local Factory_InvokeProtocol     = arch.TypedInvokeProtocolType{
                corba_protocol.InvokeProtocol
              }
local Factory_ListenProtocol     = arch.TypedListenProtocolType{
                corba_protocol.ListenProtocol
              }

local Factory_PassiveChannel     = arch.ChannelFactoryType{
                channel_factory.PassiveChannelFactory
              }
local Factory_ActiveChannel      = arch.ChannelFactoryType{
                channel_factory.ActiveChannelFactory
              }

local Factory_Reference          = arch.ReferenceResolverType{
                corba_reference
              }

local Factory_Manager            = arch.TypeManagerType{ manager }

local Factory_ClientBroker       = arch.ClientBrokerType{ client_broker }
local Factory_Proxy              = arch.TypedProxyFactoryType{ proxy }
```

```
41
42 local Factory_Dispatcher        = arch.TypedDispatcherType{ dispatcher }
43 local Factory_ServerBroker      = arch.ServerBrokerType{ server_broker }
44 local Factory_Acceptor          = arch.AcceptorType{ access_point }
45
46 ------------------------------------------
47 myCodec                        = Factory_Codec()
48 myInvokeProtocol               = Factory_InvokeProtocol()
49 myListenProtocol               = Factory_ListenProtocol()
50 myReferenceResolver            = Factory_Reference()
51 myAcceptor                     = Factory_Acceptor()
52
53 myClientBroker                 = Factory_ClientBroker()
54 myProxy                        = Factory_Proxy()
55 myPassiveChannelFactory        = Factory_PassiveChannel()
56 myActiveChannelFactory         = Factory_ActiveChannel()
57 myDispatcher                   = Factory_Dispatcher()
58 myServerBroker                 = Factory_ServerBroker()
59 myManager                      = Factory_Manager()
60 ------------------------------------------
61 myInvokeProtocol._cache_enabled = true
62 myInvokeProtocol.codec         = myCodec.codec
63 myInvokeProtocol.channels      = myActiveChannelFactory.factory
64
65 myListenProtocol.codec         = myCodec.codec
66 myListenProtocol.channels      = myPassiveChannelFactory.factory
67 myListenProtocol.interfaces      = myManager.registry
68
69 myReferenceResolver.codec      = myCodec.codec
70
71 myClientBroker.protocol        = myInvokeProtocol.invoker
72 myClientBroker.reference       = myReferenceResolver.resolver
73 myClientBroker.factory         = myProxy.proxies
74
75 myProxy.interfaces             = myManager.registry
76
77 myAcceptor.listener            = myListenProtocol.listener
78 myAcceptor.dispatcher          = myDispatcher.dispatcher
79
80 myDispatcher.interfaces          = myManager.registry
81
82 myServerBroker.ports["corba"]       = myAcceptor.manager
83 myServerBroker.objectmap       = myDispatcher.registry
84 myServerBroker.reference["corba"]      = myReferenceResolver.resolver
85
86 myPassiveChannelFactory.luasocket = require "oil.socket"
87 myActiveChannelFactory.luasocket  = require "oil.socket"
```

Listagem C.1: Configuração do OiL usando o protocolo CORBA.

Configuração simples usando LuaDummy

```
1 module "oil.configs.DummySimple"
2 ------------------------------------------
3 -- binding components
4 local arch = require "oil.arch.comm"
5
```

```
6  local dummy_codec          = require "oil.dummy.Codec"
7  local dummy_protocol       = require "oil.dummy.Protocol"
8  local dummy_reference      = require "oil.dummy.reference"
9
10 local proxy                = require "oil.dummy.proxy"
11
12 local client_broker        = require "oil.ClientBroker"
13 local server_broker        = require "oil.ServerBroker"
14 local channel_factory      = require "oil.ChannelFactory"
15 local dispatcher           = require "oil.dummy.Dispatcher"
16 local acceptor             = require "oil.SimpleAcceptor"
17
18 ----------------------------------------
19 local Factory_Codec              = arch.CodecType{ dummy_codec }
20 local Factory_InvokeProtocol     = arch.InvokeProtocolType{
21                 dummy_protocol.InvokeProtocol
22               }
23 local Factory_ListenProtocol     = arch.ListenProtocolType{
24                 dummy_protocol.ListenProtocol
25               }
26 local Factory_PassiveChannel     = arch.ChannelFactoryType{
27                  channel_factory.PassiveChannelFactory
28               }
29 local Factory_ActiveChannel      = arch.ChannelFactoryType{
30                 channel_factory.ActiveChannelFactory
31               }
32
33 local Factory_Reference = arch.ReferenceResolverType{ dummy_reference }
34
35 local Factory_ClientBroker       = arch.ClientBrokerType{ client_broker }
36 local Factory_Proxy              = arch.ProxyFactoryType{ proxy }
37
38 local Factory_Dispatcher         = arch.DispatcherType{ dispatcher }
39 local Factory_ServerBroker       = arch.ServerBrokerType{ server_broker }
40 local Factory_Acceptor           = arch.AcceptorType{ acceptor }
41
42 ----------------------------------------
43
44 myCodec = Factory_Codec()
45 myInvokeProtocol = Factory_InvokeProtocol()
46 myListenProtocol = Factory_ListenProtocol()
47 myReferenceResolver = Factory_Reference()
48 myAcceptor = Factory_Acceptor()
49
50 myClientBroker = Factory_ClientBroker()
51 myProxy = Factory_Proxy()
52 myPassiveChannelFactory = Factory_PassiveChannel()
53 myActiveChannelFactory = Factory_ActiveChannel()
54 myDispatcher = Factory_Dispatcher()
55 myServerBroker = Factory_ServerBroker()
56
57 ----------------------------------------
58 myInvokeProtocol.codec          = myCodec.codec
59 myInvokeProtocol.channels       = myActiveChannelFactory.factory
60
61 myListenProtocol.codec          = myCodec.codec
62 myListenProtocol.channels       = myPassiveChannelFactory.factory
63
64 myReferenceResolver.codec         = myCodec.codec
```

```
65
66 myClientBroker.protocol        = myInvokeProtocol.invoker
67 myClientBroker.reference = myReferenceResolver.resolver
68 myClientBroker.factory = myProxy.proxies
69
70 myAcceptor.listener       = myListenProtocol.listener
71 myAcceptor.dispatcher       = myDispatcher.dispatcher
72
73 myServerBroker.ports["dummy"] = myAcceptor.manager
74 myServerBroker.objectmap = myDispatcher.registry
75 myServerBroker.reference["dummy"] = myReferenceResolver.resolver
76
77 myActiveChannelFactory.luasocket = require "oil.socket"
78 myPassiveChannelFactory.luasocket = require "oil.socket"
```

Listagem C.2: Configuração do OiL usando o protocolo LuaDummy.

Configuração usando SSL como transporte

```
1  module "oil.configs.CORBASslSimple"
2  ------------------------------------------------
3  local arch = require "oil.arch.comm"
4
5  local corba_codec          = require "oil.corba.Codec"
6  local corba_protocol       = require "oil.corba.Protocol"
7  local corba_reference      = require "oil.corba.reference"
8
9  local proxy              = require "oil.corba.proxy"
10 local client_broker      = require "oil.ClientBroker"
11 local server_broker      = require "oil.ServerBroker"
12 local channel_factory    = require "oil.ChannelFactory"
13 local dispatcher         = require "oil.corba.SimpleDispatcher"
14 local manager            = require "oil.ir"
15 local access_point       = require "oil.SimpleAcceptor"
16
17 ------------------------------------------------
18 local Factory_Codec              = arch.CodecType{ corba_codec }
19 local Factory_InvokeProtocol     = arch.TypedInvokeProtocolType{
20          corba_protocol.InvokeProtocol
21        }
22 local Factory_ListenProtocol     = arch.TypedListenProtocolType{
23          corba_protocol.ListenProtocol
24        }
25
26 local Factory_PassiveChannel     = arch.ChannelFactoryType{
27          channel_factory.PassiveChannelFactory
28        }
29 local Factory_ActiveChannel      = arch.ChannelFactoryType{
30          channel_factory.ActiveChannelFactory
31        }
32
33 local Factory_Reference          = arch.ReferenceResolverType{
34          corba_reference
35        }
36
37 local Factory_Manager            = arch.TypeManagerType{ manager }
38
```

```
39 local Factory_ClientBroker      = arch.ClientBrokerType{ client_broker }
40 local Factory_Proxy             = arch.TypedProxyFactoryType{ proxy }
41
42 local Factory_Dispatcher        = arch.TypedDispatcherType{ dispatcher }
43 local Factory_ServerBroker      = arch.ServerBrokerType{ server_broker }
44 local Factory_Acceptor          = arch.AcceptorType{ access_point }
45
46 local Factory_Scheduler         = arch.SchedulerType{ scheduler }
47 ----------------------------------------
48 myCodec                        = Factory_Codec()
49 myInvokeProtocol               = Factory_InvokeProtocol()
50 myListenProtocol               = Factory_ListenProtocol()
51 myReferenceResolver            = Factory_Reference()
52 myAcceptor                     = Factory_Acceptor()
53
54 myClientBroker                 = Factory_ClientBroker()
55 myProxy                        = Factory_Proxy()
56 myPassiveChannelFactory        = Factory_PassiveChannel()
57 myActiveChannelFactory         = Factory_ActiveChannel()
58 myDispatcher                   = Factory_Dispatcher()
59 myServerBroker                 = Factory_ServerBroker()
60 myManager                      = Factory_Manager()
61
62 myScheduler = Factory_Scheduler()
63 ----------------------------------------
64 myInvokeProtocol.codec         = myCodec.codec
65 myInvokeProtocol.channels      = myActiveChannelFactory.factory
66
67 myListenProtocol.codec         = myCodec.codec
68 myListenProtocol.channels      = myPassiveChannelFactory.factory
69 myListenProtocol.interfaces       = myManager.registry
70
71 myReferenceResolver.codec      = myCodec.codec
72
73 myClientBroker.protocol        = myInvokeProtocol.invoker
74 myClientBroker.reference       = myReferenceResolver.resolver
75 myClientBroker.factory         = myProxy.proxies
76
77 myProxy.interfaces             = myManager.registry
78
79 myAcceptor.listener            = myListenProtocol.listener
80 myAcceptor.dispatcher          = myDispatcher.dispatcher
81
82 myDispatcher.interfaces           = myManager.registry
83
84 myServerBroker.ports["corba"]        = myAcceptor.manager
85 myServerBroker.objectmap       = myDispatcher.registry
86 myServerBroker.reference["corba"]       = myReferenceResolver.resolver
87
88 local socket = require "oil.socket"
89 socket.server_params = {
90     key = "certs/serverkey.pem",
91     cert = "certs/server.pem",
92     cafile = "certs/rootA.pem",
93     method = "SSLv3",
94     verify = {"peer", "fail_if_no_peer_cert"},
95     options = {"all", "no_sslv2"},
96 }
97 socket.client_params = {
```

```
98      key = "certs/clientkey.pem",
99      cert = "certs/client.pem",
100     cafile = "certs/rootA.pem",
101     method = "SSLv3",
102     verify = {"peer", "fail_if_no_peer_cert"},
103     options = {"all", "no_sslv2"},
104 }
105 myPassiveChannelFactory.luasocket = socket
106 myActiveChannelFactory.luasocket  = socket
```

Listagem C.3: Configuração do OiL usando o transporte SSL.

Configuração usando um escalonador no servidor

```
1  module "oil.configs.CORBAConcurrentServer"
2  ------------------------------------------------
3  local arch = require "oil.arch.comm"
4
5  local scheduler         = require "oil.scheduler"
6
7  local corba_codec         = require "oil.corba.Codec"
8  local corba_protocol      = require "oil.corba.Protocol"
9  local corba_reference     = require "oil.corba.reference"
10
11 local proxy             = require "oil.corba.proxy"
12 local client_broker     = require "oil.ClientBroker"
13 local server_broker     = require "oil.ServerBroker"
14 local channel_factory   = require "oil.ChannelFactory"
15 local dispatcher         = require "oil.corba.ConcurrentDispatcher"
16 local manager            = require "oil.ir"
17 local access_point       = require "oil.ConcurrentAcceptor"
18
19 ----------------------------------------
20 local Factory_Codec              = arch.CodecType{ corba_codec }
21 local Factory_InvokeProtocol     = arch.TypedInvokeProtocolType{
22         corba_protocol.InvokeProtocol
23       }
24 local Factory_ListenProtocol     = arch.TypedListenProtocolType{
25         corba_protocol.ListenProtocol
26       }
27
28 local Factory_PassiveChannel     = arch.ChannelFactoryType{
29         channel_factory.PassiveChannelFactory
30       }
31 local Factory_ActiveChannel      = arch.ChannelFactoryType{
32         channel_factory.ActiveChannelFactory
33       }
34
35 local Factory_Reference          = arch.ReferenceResolverType{
36         corba_reference
37       }
38
39 local Factory_Manager            = arch.TypeManagerType{ manager }
40
41 local Factory_ClientBroker       = arch.ClientBrokerType{ client_broker }
42 local Factory_Proxy              = arch.TypedProxyFactoryType{ proxy }
43
```

```
44 local Factory_Dispatcher        = arch.TypedDispatcherType{ dispatcher }
45 local Factory_ServerBroker      = arch.ServerBrokerType{ server_broker }
46 local Factory_Acceptor          = arch.AcceptorType{ access_point }
47
48 local Factory_Scheduler         = arch.SchedulerType{ scheduler }
49
50 ----------------------------------------
51 myCodec                        = Factory_Codec()
52 myInvokeProtocol               = Factory_InvokeProtocol()
53 myListenProtocol               = Factory_ListenProtocol()
54 myReferenceResolver            = Factory_Reference()
55 myAcceptor                     = Factory_Acceptor()
56
57 myClientBroker                 = Factory_ClientBroker()
58 myProxy                        = Factory_Proxy()
59 myPassiveChannelFactory        = Factory_PassiveChannel()
60 myActiveChannelFactory         = Factory_ActiveChannel()
61 myDispatcher                   = Factory_Dispatcher()
62 myServerBroker                 = Factory_ServerBroker()
63 myManager                      = Factory_Manager()
64
65 myScheduler = Factory_Scheduler()
66
67 ----------------------------------------
68 myInvokeProtocol.codec         = myCodec.codec
69 myInvokeProtocol.channels      = myActiveChannelFactory.factory
70
71 myListenProtocol.codec         = myCodec.codec
72 myListenProtocol.channels      = myPassiveChannelFactory.factory
73 myListenProtocol.interfaces       = myManager.registry
74
75 myReferenceResolver.codec      = myCodec.codec
76
77 myClientBroker.protocol        = myInvokeProtocol.invoker
78 myClientBroker.reference       = myReferenceResolver.resolver
79 myClientBroker.factory         = myProxy.proxies
80
81 myProxy.interfaces             = myManager.registry
82
83 myAcceptor.listener            = myListenProtocol.listener
84 myAcceptor.dispatcher          = myDispatcher.dispatcher
85 myAcceptor.tasks               = myScheduler.threads
86
87 myDispatcher.tasks             = myScheduler.threads
88 myDispatcher.interfaces           = myManager.registry
89
90 myServerBroker.ports["corba"]        = myAcceptor.manager
91 myServerBroker.objectmap       = myDispatcher.registry
92 myServerBroker.reference["corba"]       = myReferenceResolver.resolver
93
94 myPassiveChannelFactory.luasocket = myScheduler.socket
95 myActiveChannelFactory.luasocket  = require "oil.socket"
```

Listagem C.4: Configuração do OiL usando o escalonador.

Configuração de um servidor respondendo dois protocolos

```
module "oil.configs.CORBAServerDummyServer"
------------------------------------------------
local arch = require "oil.arch.comm"

local scheduler           = require "oil.scheduler"
local corba_codec         = require "oil.corba.Codec"
local corba_protocol      = require "oil.corba.Protocol"
local corba_reference     = require "oil.corba.reference"
local dummy_codec         = require "oil.dummy.Codec"
local dummy_protocol      = require "oil.dummy.Protocol"
local dummy_reference     = require "oil.dummy.reference"

local server_broker     = require "oil.ServerBroker"
local channel_factory   = require "oil.ChannelFactory"
local corba_dispatcher  = require "oil.corba.SimpleDispatcher"
local dummy_dispatcher  = require "oil.dummy.Dispatcher"
local manager           = require "oil.ir"
local access_point      = require "oil.ConcurrentAcceptor"

------------------------------------------------
local Factory_CorbaCodec              = arch.CodecType{ corba_codec }
local Factory_DummyCodec              = arch.CodecType{ dummy_codec }
local Factory_CorbaListenProtocol = arch.TypedListenProtocolType{
      corba_protocol.ListenProtocol
    }
local Factory_DummyListenProtocol = arch.ListenProtocolType{
      dummy_protocol.ListenProtocol
    }

local Factory_PassiveChannel    = arch.ChannelFactoryType{
      channel_factory.PassiveChannelFactory
    }

local Factory_CorbaReference          = arch.ReferenceResolverType{
      corba_reference
    }
local Factory_DummyReference          = arch.ReferenceResolverType{
      dummy_reference
    }

local Factory_Manager           = arch.TypeManagerType{ manager }

local Factory_CorbaDispatcher   = arch.TypedDispatcherType{
      corba_dispatcher
    }
local Factory_ServerBroker      = arch.ServerBrokerType{ server_broker }
local Factory_Acceptor          = arch.AcceptorType{ access_point }
local Factory_Scheduler         = arch.SchedulerType{ scheduler }

------------------------------------------------
myCodecCorba                    = Factory_CorbaCodec()
myCodecDummy                    = Factory_DummyCodec()
myListenProtocolCorba           = Factory_CorbaListenProtocol()
myListenProtocolDummy           = Factory_DummyListenProtocol()
myAcceptorCorba                 = Factory_Acceptor()
myAcceptorDummy                 = Factory_Acceptor()

myReferenceResolverCorba        = Factory_CorbaReference()
myReferenceResolverDummy        = Factory_DummyReference()
```

```
60
61 myPassiveChannelFactory         = Factory_PassiveChannel()
62 myDispatcher                    = Factory_CorbaDispatcher()
63 myServerBroker                  = Factory_ServerBroker()
64 myManager                       = Factory_Manager()
65
66 myScheduler                     = Factory_Scheduler()
67 ----------------------------------------
68 myListenProtocolCorba.codec     = myCodecCorba.codec
69 myListenProtocolCorba.channels = myPassiveChannelFactory.factory
70 myListenProtocolCorba.interfaces  = myManager.mapping
71
72 myListenProtocolDummy.codec     = myCodecDummy.codec
73 myListenProtocolDummy.channels = myPassiveChannelFactory.factory
74
75 myReferenceResolverCorba.codec = myCodecCorba.codec
76
77 myAcceptorCorba.listener        = myListenProtocolCorba.listener
78 myAcceptorCorba.dispatcher      = myDispatcher.dispatcher
79 myAcceptorCorba.tasks           = myScheduler.threads
80 myAcceptorDummy.listener        = myListenProtocolDummy.listener
81 myAcceptorDummy.dispatcher      = myDispatcher.dispatcher
82 myAcceptorDummy.tasks           = myScheduler.threads
83
84 myDispatcher.interfaces            = myManager.registry
85
86 myServerBroker.ports["corba"]   = myAcceptorCorba.manager
87 myServerBroker.ports["dummy"]   = myAcceptorDummy.manager
88 myServerBroker.objectmap        = myDispatcher.registry
89 myServerBroker.reference["corba"] = myReferenceResolverCorba.resolver
90 myServerBroker.reference["dummy"] = myReferenceResolverDummy.resolver
91
92 myPassiveChannelFactory.luasocket = myScheduler.socket
```

Listagem C.5: Configuração do OiL respondendo dois protocolos.

Configuração de uma ponte entre protocolos

```
1  module "oil.configs.CORBAClientDummyServer"
2  ----------------------------------------------------
3  local arch = require "oil.arch.comm"
4
5  local corba_codec          = require "oil.corba.Codec"
6  local corba_protocol       = require "oil.corba.Protocol"
7  local corba_reference      = require "oil.corba.reference"
8  local dummy_codec          = require "oil.dummy.Codec"
9  local dummy_protocol       = require "oil.dummy.Protocol"
10 local dummy_reference      = require "oil.dummy.reference"
11
12 local proxy              = require "oil.corba.proxy"
13 local dispatcher           = require "oil.dummy.Dispatcher"
14
15 local client_broker      = require "oil.ClientBroker"
16 local server_broker      = require "oil.ServerBroker"
17 local channel_factory    = require "oil.ChannelFactory"
18 local manager            = require "oil.ir"
19 local access_point       = require "oil.SimpleAcceptor"
```

```
20
21 local Factory_CorbaCodec              = arch.CodecType{ corba_codec }
22 local Factory_DummyCodec              = arch.CodecType{ dummy_codec }
23 local Factory_InvokeProtocol    = arch.TypedInvokeProtocolType{
24           corba_protocol.InvokeProtocol
25         }
26 local Factory_ListenProtocol    = arch.ListenProtocolType{
27           dummy_protocol.ListenProtocol
28         }
29
30 local Factory_PassiveChannel    = arch.ChannelFactoryType{
31           channel_factory.PassiveChannelFactory
32         }
33 local Factory_ActiveChannel     = arch.ChannelFactoryType{
34           channel_factory.ActiveChannelFactory
35         }
36
37 local Factory_CorbaReference          = arch.ReferenceResolverType{
38           corba_reference
39         }
40 local Factory_DummyReference          = arch.ReferenceResolverType{
41           dummy_reference
42         }
43
44 local Factory_Manager            = arch.TypeManagerType{ manager }
45
46 local Factory_ClientBroker       = arch.ClientBrokerType{ client_broker }
47 local Factory_Proxy              = arch.TypedProxyFactoryType{ proxy }
48
49 local Factory_Dispatcher         = arch.DispatcherType{ dispatcher }
50 local Factory_ServerBroker       = arch.ServerBrokerType{ server_broker }
51 local Factory_Acceptor           = arch.AcceptorType{ access_point }
52 ----------------------------------------
53
54 myCorbaCodec = Factory_CorbaCodec()
55 myDummyCodec = Factory_DummyCodec()
56 myInvokeProtocol = Factory_InvokeProtocol()
57 myListenProtocol = Factory_ListenProtocol()
58 myCorbaReferenceResolver = Factory_CorbaReference()
59 myDummyReferenceResolver = Factory_DummyReference()
60 myAcceptor = Factory_Acceptor()
61
62 myClientBroker = Factory_ClientBroker()
63 myProxy = Factory_Proxy()
64 myPassiveChannelFactory = Factory_PassiveChannel()
65 myActiveChannelFactory = Factory_ActiveChannel()
66 myDispatcher = Factory_Dispatcher()
67 myServerBroker = Factory_ServerBroker()
68 myManager = Factory_Manager()
69
70 ----------------------------------------
71 myInvokeProtocol.codec          = myCorbaCodec.codec
72 myInvokeProtocol.channels       = myActiveChannelFactory.factory
73
74 myListenProtocol.codec          = myDummyCodec.codec
75 myListenProtocol.channels       = myPassiveChannelFactory.factory
76
77 myCorbaReferenceResolver.codec         = myCorbaCodec.codec
78
```

```
79 myClientBroker.protocol        = myInvokeProtocol.invoker
80 myClientBroker.reference = myCorbaReferenceResolver.resolver
81 myClientBroker.factory = myProxy.proxies
82
83 myProxy.interfaces = myManager.registry
84
85 myAcceptor.listener       = myListenProtocol.listener
86 myAcceptor.dispatcher       = myDispatcher.dispatcher
87
88 myServerBroker.ports["dummy"] = myAcceptor.manager
89 myServerBroker.objectmap = myDispatcher.registry
90 myServerBroker.reference["dummy"] = myDummyReferenceResolver.resolver
91
92 myPassiveChannelFactory.luasocket = require "oil.socket"
93 myActiveChannelFactory.luasocket  = require "oil.socket"
```

Listagem C.6: Configuração do OiL fazendo uma ponte entre dois protocolos.

D
Testes de desempenho – código fonte

Nas próximas seções, apresentamos o código fonte usado para os testes de desempenho da seção 5.2. O cliente recebe o tipo de teste e o número de iterações do teste como parâmetros.

Interface IDL

```
struct StructOne {
        short number;
        string txt;
};
struct StructTwo {
        string name;
        string email;
};
typedef sequence<StructTwo> StructSeq;
struct BigStruct {
        string title;
        StructSeql seq_structs;
};
interface PerformanceTest {
  string request1(in string str);
  void request2(in StructOne struct1, in StructOne struct2);
  void request3(in BigStruct struct1);
};
```

Listagem D.1: Interface em IDL dos testes de desempenho.

Cliente

```
require "oil"

oil.Config.flavor = "CORBASimple"
oil.init()
------------------------------------------------------------
-- Load the interface from IDL file ------------------------
oil.loadidlfile("test.idl")

------------------------------------------------------------
-- Get object reference from file --------------------------

local ior
local file = io.open("test.ior")
```

```
14 if file then
15   ior = file:read("*a")
16   file:close()
17 else
18   print "unable to read IOR from file 'hello.ior'"
19   os.exit(1)
20 end
21
22 ----------------------------------------------------------------
23 -- Create an object proxy for the supplied interface ----------
24
25 local proxy = oil.newproxy(ior, "PerformanceTest")
26
27
28 ----------------------------------------------------------------
29 -- Helper Functions ------------------------------------------
30 Titles = {
31         "Guidelines for Welfare Analysis",
32         "Social Assistance in Albania",
33         "Poverty Lines in Theory and Practice",
34         "The Role of the Private Sector in Education in Vietnam",
35 ...
36 }
37 Authors = {
38         { name ="Afvpmj Xtewsnhacj Dz" , email ="daaj@opperl.jbp.uu" },
39         { name ="Nwsy Ydmgwfu Vzpkgh"  , email ="ajcrbax@hpr.vsm.ft" },
40         { name ="Mytcp Kpojdflun Zie"  , email ="s@dydxsh.lbz.rd"     },
41         { name ="Wpeevmen Thorts Mj"   , email ="anrlc@imil.hzh.to"  },
42 ...
43 }
44 function createsequence()
45         local myseq = {}
46         for i=1, math.random(100) do
47                 table.insert(myseq, Authors[math.random(#Authors)])
48         end
49         return myseq
50 end
51
52 ----------------------------------------------------------------
53 -- Tests -----------------------------------------------------
54 local tests = {
55   test1 = function()
56     for i = 1, arg[2] do
57       hello:say_hello_to("world")
58     end
59   end,
60   test2 = function()
61     local tbl1 = {number=10,
62       txt = "The quick brown fox jumps over the lazy dog"
63     }
64     local tbl2 = {number=20, txt = "Small string"}
65     for i = 1, arg[2] do
66       hello:request2(tbl1, tbl2)
67     end
68   end,
69   test3 = function()
70     for i = 1, arg[2] do
71       hello:submit{
72               title = Titles[math.random(table.getn(Titles))];
```

```
73                seq_structs = createsequence();
74        }
75      end
76    end,
77 }
```

Listagem D.2: Código fonte do cliente usado nos testes de desempenho.

Servidor

```
1  -- Load OiL package
2  require "oil"
3
4  oil.Config.flavor = "CORBASimple"
5  oil.init()
6
7  -- Load the interface from IDL file
8  oil.loadidlfile("test.idl")
9
10 -- Create object implementation
11 local servant_impl = { count = 0, }
12 function servant_impl:request1(str)
13   self.count = self.count + 1
14   local msg = "Hello " .. name .. "! (".. self.count.." times)"
15   print(msg)
16   return msg
17 end
18
19 function servant_impl:request2(struct1, struct2)
20   for k,v in pairs(struct1) do
21     print(k, v)
22   end
23 end
24
25 function servant_impl:request3(struct)
26   for k,v in ipairs(struct.seq_structs) do
27     -- do nothing
28   end
29 end
30
31 -- Create CORBA object
32 servant = oil.newobject(servant_impl, "PerformanceTest")
33
34 local file = io.open("test.ior", "w")
35 if file then
36   -- Write object ref. into file
37   file:write(oil.getreference(servant))
38   file:close()
39 else
40   -- Show object ref. on screen
41   print(oil.getreference(servant))
42 end
43
44 -- Start ORB main loop
45 oil.run()
```

Listagem D.3: Código fonte do servidor usado nos testes de desempenho.