7

Referências Bibliográficas

[1] COULSON, G.; BLAIR, G. S.; CLARKE, M. ; PARLAVANTZAS, N.. **The design of a configurable and reconfigurable middleware platform**. Distributed Computing, 15(2):109–126, 2002.

[2] HENNING, M.. **A new approach to object-oriented middleware**. IEEE Internet Computing, 8(4):66–75, November 2004.

[3] GRACE, P.; COULSON, G.; BLAIR, G. ; PORTER, B.. **Deep middleware for the divergent grid**. Middleware 2005, p. 334–353, 2005.

[4] PFISTER, C.; SZYPERSKI, C.. **Why objects are not enough**. Em: PROCEEDINGS, INTERNATIONAL COMPONENT USERS CONFERENCE, Munich, Germany, 1996. SIGS.

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

[6] STAA, A. V.. **Programação Modular: Desenvolvendo Programas Complexos de Forma Organizada e Segura**. Campus, Rio de Janeiro, Brasil, 2000.

[7] OMG: OBJECT MANAGEMENT GROUP. **CORBA Component Model**. `http://www.omg.org/technology/documents/formal/components.htm`, April 2006.

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

[9] KON, F.; MARQUES, J. R.; YAMANE, T.; CAMPBELL, R. H. ; MICKUNAS, M. D.. **Design, implementation, and performance of an**

automatic configuration service for distributed component systems: Research articles. Softw. Pract. Exper., 35(7):667–703, 2005.

[10] WALKER, R. J.; BANIASSAD, E. L. A. ; MURPHY, G. C.. **An initial assessment of aspect-oriented programming**. Em: ICSE '99: PROCEEDINGS OF THE 21ST INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, p. 120–130, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[11] KICZALES, G.; LAMPING, J.; MENHDHEKAR, A.; MAEDA, C.; LOPES, C.; LOINGTIER, J.-M. ; IRWIN, J.. **Aspect-oriented programming**. Em: Akşit, M.; Matsuoka, S., editors, PROCEEDINGS EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, volume 1241, p. 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[12] ELRAD, T.; AKSIT, M.; KICZALES, G.; LIEBERHERR, K. ; OSSHER, H.. **Discussing aspects of AOP**. Commun. ACM, 44(10):33–38, outubro 2001.

[13] DE MOURA, A. L.; URURAHY, C.; CERQUEIRA, R. ; RODRIGUEZ, N.. **Dynamic support for distributed auto-adaptive applications**. Em: PROCEEDINGS OF AOPDCS'02 – WORKSHOP ON ASPECT ORIENTED PROGRAMMING FOR DISTRIBUTED COMPUTING SYSTEMS (EM CONJUNTO COM IEEE ICDCS 2002), p. 451–456, Viena, Áustria, julho 2002.

[14] FLEURY, M.; REVERBEL, F.. **The JBoss extensible server**. Em: Endler, M.; Schmidt, D., editors, MIDDLEWARE 2003 — ACM/IFIP/USENIX INTERNATIONAL MIDDLEWARE CONFERENCE, volume 2672 de **LNCS**, p. 344–373. Springer-Verlag, 2003.

[15] MICROSOFT CORPORATION. **COM - Component Object Model**. `http://www.microsoft.com/com/`.

[16] **CORBA - Common Object Request Broker Architecture**. `http://www.omg.org/gettingstarted/corbafaq.htm/`.

[17] FOSTER, I.; KESSELMAN, C.. **Globus: A metacomputing infrastructure toolkit**. International Journal of Supercomputer Applications, 11:115–128, 1997.

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

[19] KON, F.; ROMÁN, M.; LIU, P.; MAO, J.; YAMANE, T.; CLAUDIO MAGALH A. ; CAMPBELL, R. H.. **Monitoring, security, and dynamic configuration with the dynamictao reflective ORB**. Em: MIDDLEWARE '00: IFIP/ACM INTERNATIONAL CONFERENCE ON DISTRIBUTED SYSTEMS PLATFORMS, p. 121–143, Secaucus, NJ, USA, 2000. Springer-Verlag New York, Inc.

[20] LINDFORS, J.; FLEURY, M. ; GROUP, T. J.. **JMX: Managing J2EE With Java Management Extensions**. Sams Publishing, 2002.

[21] 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: COMPSAC '03: PROCEEDINGS OF THE 27TH ANNUAL INTERNATIONAL CONFERENCE ON COMPUTER SOFTWARE AND APPLICATIONS, p. 400, Washington, DC, USA, 2003. IEEE Computer Society.

[22] LONG, B.; STROOPER, P.. **A case study in testing distributed systems**. doa, 00:0020, 2001.

[23] TSAI, W.; YU, L. ; SAIMI, A.. **Scenario-based object-oriented test frameworks for testing distributed systems**. ftdcs, 00:288, 2003.

[24] WALTER, T.; SCHIEFERDECKER, I. ; GRABOWSKI, J.. **1 test architectures for distributed systems- state of the art and beyond**.

[25] **Tecgraf**. `http://www.tecgraf.puc-rio.br/`.

[26] **Pontifícia universidade católica do rio de janeiro**. `http://www.puc-rio.br/`.

[27] IERUSALIMSCHY, R.; CELES, W. ; DE FIGUEIREDO, L. H.. **Lua - the programming language**. `http://www.lua.org/`.

[28] IERUSALIMSCHY, R.. **Programming in Lua**. Lua.org, 2003.

[29] GROUP OF DISTRIBUTED SYSTEMS - PUC-RIO. **OiL - The Lua Object Request Broker**. `http://oil.luaforge.net/`.

[30] **OMG - the object management group**. `http://www.omg.org/`.

[31] NARDI, A. R.. **Componentes CORBA**. Dissertação de mestrado, Universidade de São Paulo, 2003.

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

[33] SUN MICROSYSTEMS. **Enterprise Java Beans**. `http://java.sun.com/products/ejb/docs.html`.

[34] OBJECTWEB. **OpenCCM**. `http://openccm.objectweb.org/`.

[35] MAIA, R. F.. **Um framework para adaptação dinâmica de sistemas baseados em componentes distribuídos**. Dissertação de mestrado, Pontifícia Universidade Católica do Rio de Janeiro, 2004.

[36] CRAWFORD, D.. **Special issue on aspect-oriented programming**. Communications of the ACM, 44(10), 2001.

[37] CAI, W.; COULSON, G.; GRACE, P.; BLAIR, G.; MATHY, L. ; YEUNG, W.. **The Gridkit Distributed Resource Management Framework**. Em: Sloot, P. M. A.; Hoekstra, A. G.; Priol, T.; Reinefeld, A. ; Bubak, M., editors, PROCEEDINGS OF THE EUROPEAN GRID CONFERENCE, volume 3470 de **Lecture Notes in Computer Science**, p. 786 – 795, Amsterdam, June 2005. Springer Berlin / Heidelberg.

[38] COULSON, G.; GRACE, P.; BLAIR, G.; CAI, W.; COOPER, C.; DUCE, D.; MATHY, L.; YEUNG, W. K.; PORTER, B.; SAGAR, M. ; LI, W.. **A component-based middleware framework for configurable and reconfigurable grid computing: Research articles**. Concurr. Comput. : Pract. Exper., 18(8):865–874, 2006.

[39] CLARKE, M.; BLAIR, G. S.; COULSON, G. ; PARLAVANTZAS, N.. **An efficient component model for the construction of adaptive middleware**. Em: MIDDLEWARE '01: PROCEEDINGS OF THE IFIP/ACM INTERNATIONAL CONFERENCE ON DISTRIBUTED SYSTEMS PLATFORMS HEIDELBERG, p. 160–178, London, UK, 2001. Springer-Verlag.

[40] COULSON, G.; BLAIR, G.; GRACE, P.; JOOLIA, A.; LEE, K. ; UEYAMA, J.. **A component model for building systems software**. Em: IN PROCEEDINGS OF IASTED SOFTWARE ENGINEERING AND APPLICATIONS (SEA'04, 2004.

[41] GRACE, P.; COULSON, G.; BLAIR, G.; MATHY, L.; YEUNG, W. K.; CAI, W.; DUCE, D. ; COOPER, C.. **Gridkit: Pluggable overlay networks for grid computing**. p. 1463–1481, 2004.

[42] **MWLab - Middleware Laboratory, PUC-Rio**. http://mwlab.tecgraf.puc-rio.br/MWLab/.

[43] GROUP OF DISTRIBUTED SYSTEMS - PUC-RIO. **LOOP - Lua Object-Oriented Programming**. http://loop.luaforge.net/.

[44] MIDDLEWARE LABORATORY, PUC-RIO. **SCS - Software Component System**. http://www.tecgraf.puc-rio.br/~scorrea/scs/.

[45] HENNING, M.. **The rise and fall of CORBA**. ACM Queue: Component Technologies, 4(5), 2006.

[46] MIKE PALL. **LuaJIT - Lua Just-In-Time Compiler**. http://luajit.org/.

[47] MASCARENHAS, F.; IERUSALIMSCHY, R.. **Efficient compilation of Lua for the CLR**. Em: SAC '08: PROCEEDINGS OF THE 2008 ACM SYMPOSIUM ON APPLIED COMPUTING, p. 217–221, New York, NY, USA, 2008. ACM.

[48] LUA-USERS WIKI. **Lua Garbage Collection Tutorial**. http://lua-users.org/wiki/GarbageCollectionTutorial, 2007.

[49] OMG. **CORBA Event Service**. http://www.omg.org/technology/documents/formal/event_service.htm/.

[50] OSGI ALLIANCE. **RFC-0112 Bundle Repository**. http://bundles.osgi.org/rfc-0112_BundleRepository.pdf, 2006.

[51] **Lua Rocks: Versioned Components For Lua**. http://www.luarocks.org/.

Apêndice A

Arquivos IDL Completos

Listing A.1: scs.idl

```
1  #ifndef SCS_IDL
2  #define SCS_IDL
3
4
5  module scs {
6    module core {
7      exception StartupFailed {};
8      exception ShutdownFailed {};
9      exception InvalidName {
10       string name;
11     };
12     exception InvalidConnection {};
13     exception AlreadyConnected {};
14     exception ExceededConnectionLimit {};
15     exception NoConnection {};
16
17
18     typedef unsigned long ConnectionId;
19     typedef sequence<string> NameList;
20
21     struct FacetDescription {
22       string name;
23       string interface_name;
24       Object facet_ref;
25     };
26     typedef sequence<FacetDescription> FacetDescriptions;
27
28     struct ConnectionDescription {
29       ConnectionId id;
30       Object objref;
31     };
```

```
32      typedef sequence<ConnectionDescription>
            ConnectionDescriptions;
33
34      struct ReceptacleDescription {
35        string name;
36        string interface_name;
37        boolean is_multiplex;
38        ConnectionDescriptions connections;
39      };
40      typedef sequence<ReceptacleDescription>
            ReceptacleDescriptions;
41
42      struct ComponentId {
43        string name;
44        unsigned long version;
45      };
46      typedef sequence<ComponentId> ComponentIdSeq;
47
48      interface IComponent {
49        void startup() raises (StartupFailed);
50        void shutdown() raises (ShutdownFailed);
51
52        Object getFacet (in string facet_interface);
53        Object getFacetByName (in string facet);
54            ComponentId getComponentId ();
55      };
56
57      interface IReceptacles {
58        ConnectionId connect (in string receptacle, in Object obj
              )
59            raises (InvalidName, InvalidConnection,
                  AlreadyConnected,
60                      ExceededConnectionLimit);
61        void disconnect (in ConnectionId id)
62            raises (InvalidConnection, NoConnection);
63        ConnectionDescriptions getConnections (in string
              receptacle)
64            raises (InvalidName);
65      };
66
67      interface IMetaInterface {
68        FacetDescriptions getFacets();
69        FacetDescriptions getFacetsByName(in NameList names)
70            raises (InvalidName);
71        ReceptacleDescriptions getReceptacles();
```

```
72        ReceptacleDescriptions getReceptaclesByName(in NameList
              names)
73            raises (InvalidName);
74      };
75    };
76  };
77
78  #endif
```

Listing A.2: deployment.idl

```
1  #ifndef DEPLOYMENT_IDL
2  #define DEPLOYMENT_IDL
3
4  #include "scs.idl"
5
6  module scs {
7
8    module container {
9
10     typedef sequence<string> StringSeq;
11     typedef sequence<octet> OctetSeq;
12     typedef sequence<core::IComponent> IComponentSeq;
13     typedef sequence<unsigned long> InterceptorIds;
14
15     struct ComponentId {
16        string name;
17        unsigned long version;
18        string platform_spec;
19     };
20     typedef sequence<ComponentId> ComponentIdSeq;
21
22     struct ComponentHandle {
23        core::IComponent cmp;
24        ComponentId id;
25        unsigned long instance_id;
26     };
27     typedef sequence<ComponentHandle> ComponentHandleSeq;
28
29     exception ComponentNotFound{};
30     exception ComponentAlreadyLoaded{};
31     exception LoadFailure{};
32     exception InterceptorNotInstalled{};
33     exception ListLockFail{};
```

```
34      exception ContainerHalted{};
35
36      interface ComponentLoader {
37         ComponentHandle load (in ComponentId id, in StringSeq
               args)
38            raises (ComponentNotFound, ComponentAlreadyLoaded,
                  LoadFailure);
39         void unload (in ComponentHandle handle)
40            raises (ComponentNotFound);
41
42         ComponentIdSeq getInstalledComponents();
43      };
44
45      interface ComponentCollection {
46        ComponentHandleSeq getComponent (in ComponentId id);
47        ComponentHandleSeq getComponents ();
48      };
49
50      interface ComponentInterception {
51        // 0 in position counts as end of the list.
52        // The same goes for positions above the list's current
             size.
53        ComponentHandle loadInterceptor (in ComponentId id, in
             StringSeq args, in unsigned long position, in string
             type)
54          raises (ListLockFail, ComponentNotFound,
                ComponentAlreadyLoaded, LoadFailure);
55        void unloadInterceptor (in ComponentHandle handle)
56          raises (InterceptorNotInstalled, ListLockFail,
                ComponentNotFound);
57        void changePosition (in unsigned long instance_id, in
             unsigned long position)
58          raises (InterceptorNotInstalled, ListLockFail,
                ComponentNotFound);
59        unsigned long getInterceptorPosition (in unsigned long
             instance_id)
60          raises (InterceptorNotInstalled, ListLockFail,
                ComponentNotFound);
61        InterceptorIds getClientInterceptorsOrder ();
62        InterceptorIds getServerInterceptorsOrder ();
63      };
64
65      interface ComponentSuspension {
66        // status:
67        // 0 means no suspension
```

```
68        // a positive number means a halt (launches exception)
69        // a negative number means a suspension (yields
              coroutines)
70        void suspend ();
71        void halt ();
72        void resume ();
73        long getStatus ();
74      };
75    };
76
77    module execution_node {
78      exception ContainerAlreadyExists{};
79      exception InvalidContainer{};
80      exception RequirementNotMet{ string reason; };
81
82      struct Property {
83         string name;
84         string value;
85         boolean read_only;
86      };
87      typedef sequence<Property> PropertySeq;
88
89      struct ContainerDescription {
90         core::IComponent container;
91         string container_name;
92         core::IComponent execution_node;
93      };
94      typedef sequence<ContainerDescription>
            ContainerDescriptionSeq;
95
96      interface ExecutionNode {
97         core::IComponent startContainer (in string
               container_name, in PropertySeq props)
98               raises (ContainerAlreadyExists);
99         void killContainer(in string container_name)
100              raises (InvalidContainer);
101        core::IComponent getContainer (in string container_name)
               ;
102        ContainerDescriptionSeq getContainers ();
103        string getName();
104     };
105
106     interface ContainerManager {
107           void registerContainer(in string name, in core::
                 IComponent ctr)
```

```
108                raises (ContainerAlreadyExists, InvalidContainer)
                       ;
109             void unregisterContainer(in string name) raises (
                    core::InvalidName);
110         };
111     };
112
113     module auxiliar{
114         exception HelpInfoNotAvailable{};
115         exception UndefinedProperty{};
116         exception ReadOnlyProperty{};
117
118         interface ComponentHelp {
119             string getHelpInfo (in container::ComponentId id)
120                 raises (container::ComponentNotFound,
                        HelpInfoNotAvailable);
121         };
122
123         interface ComponentProperties {
124             void setProperty (in execution_node::Property
                    property) raises (ReadOnlyProperty);
125             execution_node::Property getProperty (in string
                    name) raises (UndefinedProperty);
126             execution_node::PropertySeq getProperties ();
127         };
128     };
129
130 };
131
132 #endif
```

Listing A.3: repository.idl

```
1  #ifndef REPOSITORY_IDL
2  #define REPOSITORY_IDL
3
4  #include "deployment.idl"
5
6  module scs {
7      module repository {
8
9          exception ComponentAlreadyInstalled{};
10         exception ComponentNotInstalled{};
11
```

```
12          struct ComponentDescription {
13            container::ComponentId id;
14            string entry_point;
15            boolean shared;
16            string extension;
17          };
18          typedef sequence<ComponentDescription>
               ComponentDescriptionSeq;
19
20          interface ComponentRepository {
21              void install (in container::ComponentId id, in
                   string entry_point,
22                     in boolean shared, in container::OctetSeq file
                           , in string help_info, in string extension
                           )
23                     raises (ComponentAlreadyInstalled);
24
25              void uninstall (in container::ComponentId id)
26                   raises (ComponentNotInstalled);
27
28              void copy (in container::ComponentId id, in
                   ComponentRepository rep)
29                   raises (ComponentAlreadyInstalled,
                        ComponentNotInstalled);
30
31              container::OctetSeq getComponentFile (in container::
                   ComponentId id)
32                   raises (ComponentNotInstalled);
33
34              ComponentDescription getComponentDescription (in
                   container::ComponentId id)
35                   raises (ComponentNotInstalled);
36
37              ComponentDescriptionSeq  getInstalledComponents();
38           };
39       };
40  };
41  #endif
```

Listing A.4: events.idl

```
1  #ifndef EVENTS_IDL
2  #define EVENTS_IDL
3
```

```
4  #include "scs.idl"
5
6  module scs {
7    module event_service {
8       typedef any Event;
9
10      exception NameAlreadyInUse {
11         string name;
12      };
13
14      exception InvalidName {
15         string name;
16      };
17
18
19      struct ChannelDescr {
20         string name;
21         core::IComponent channel;
22      };
23      typedef sequence<ChannelDescr> ChannelDescrSeq;
24
25
26
27      interface EventSink {
28         void push (in Event ev);
29         void disconnect();
30      };
31
32
33      interface ChannelFactory {
34         core::IComponent create (in string name) raises (
               NameAlreadyInUse);
35         void destroy (in string name) raises (InvalidName);
36      };
37
38      interface ChannelCollection {
39         core::IComponent getChannel (in string name);
40         ChannelDescrSeq getAll();
41      };
42
43   };
44 };
45 #endif
```

Listing A.5: pingpong.idl

```
1  module scs{
2    module demos{
3      module pingpong {
4        interface PingPong {
5          void setId(in long identifier);
6          long getId();
7          void ping();
8          void pong();
9          void start();
10         void stop();
11       };
12     };
13   };
14 };
```

Apêndice B

Descrições de Métodos das Interfaces

B.1

Interfaces do Modelo SCS

- Faceta *IComponent*
  - *startup*(): Este método deve ser executado após a instanciação e realização de conexões de um componente. Seu objetivo é iniciar o funcionamento do componente, nos mesmos moldes de serviços Windows ou *daemons* UNIX. No caso de componentes SCS, uma tarefa comum a ser executada aqui é o teste de dependências, ou seja, a verificação de que existem facetas conectadas aos receptáculos existentes.
  - *shutdown*(): Este método serve para terminar um componente. Como componentes SCS normalmente dependem de uma entidade contêiner, a qual controlará o processo onde o componente está executando, em geral será necessário apenas liberar os recursos alocados pelo próprio componente.
  - *getFacet*(*in string facet_interface*): Retorna uma referência para a faceta da interface especificada. A referência é retornada com o tipo básico Object de CORBA, e pode ser necessário especializar o *proxy* criado no código remoto através do uso da facilidade *narrow*, provida pelos ORBs.
  - *getFacetByName*(*in string facet*): Retorna uma referência para a faceta do nome especificado. A referência é retornada com o tipo básico Object de CORBA, e pode ser necessário especializar o *proxy* criado no código remoto através do uso da facilidade *narrow*, provida pelos ORBs.

  - *getComponentId*(): Retorna a identificação do componente, que é a mesma que foi fornecida ao chamar a função newComponent() da API, composta por nome e versão do componente.

- Faceta *IReceptacles*

  - *connect*(*in string receptacle, in Object obj*): Conecta a referência para uma faceta *obj* no receptáculo de nome *receptacle*. A referência é automaticamente especializada e testada, e um número de identificação é retornado.
  - *disconnect*(*in ConnectionId id*): Desfaz uma conexão específica.
  - *getConnections*(*in string receptacle*): Retorna todas as conexões ativas do receptáculo especificado.

- Faceta *IMetaInterface*

  - *getFacets*(): Retorna as descrições de todas as facetas do componente.
  - *getFacetsByName*(*in NameList names*): Retorna as descrições de todas as facetas especificadas na lista.
  - *getReceptacles*(): Retorna as descrições de todas os receptáculos do componente.
  - *getReceptaclesByName*(*in NameList names*): Retorna as descrições de todas os receptáculos especificados na lista.

B.2
Interfaces do Nó de Execução

- Faceta *IComponent*

  - *startup*(): A versão deste método, no Nó de Execução, carrega seu arquivo de inicialização(opcional) e recupera contêineres persistidos caso existam. Caso o método seja chamado novamente durante a mesma execução, nada é feito e apenas uma mensagem de *log* é registrada.
  - *shutdown*(): Este método, no Nó de Execução, executa uma chamada *shutdown*() assíncrona em todos os seus contêineres, caso a propriedade *restart* seja falsa. Como as chamadas são assíncronas, o nó então espera 5 segundos (ou o valor da propriedade *timeout*, caso esteja definida) para que os contêineres se desregistrem. Ao fim

deste tempo, o ORB é finalizado e o processo do Nó de Execução terminado. O comportamento especificado é que os contêineres ignorem o recebimento de erros, caso não consigam contactar o Nó de Execução ao tentar se desregistrar.

- Faceta *ExecutionNode*
  - *startContainer*(*in string container_name*, *in PropertySeq props*): Inicia um novo processo contêiner, que atualmente pode ser Java ou Lua. Contêineres são identificados pelo nome, que é recebido como primeiro parâmetro. O segundo parâmetro, que é um conjunto de propriedades, atualmente serve para informar opcionalmente a linguagem (o padrão é Lua) e a máquina virtual desejada. Após iniciar o processo, o Nó de Execução aguarda 20 segundos ou o tempo definido na propriedade *timeout*. Caso o contêiner se registre neste tempo, é realizada uma chamada ao seu método *startup*().
  - *killContainer*(*in ConnectionId id*): Destrói um contêiner, através de seu *PID*, o identificador do processo fornecido pelo sistema operacional. É importante notar que não é realizada uma chamada ao método *shutdown*() do contêiner e o sistema operacional destrói o processo imediatamente, de forma que este não tenha chance de se preparar para sua finalização. Por fim, o método realiza o desregistro do contêiner em suas estruturas de dados internas.
  - *getContainer*(*in string container_name*): Fornece a referência remota para a faceta IComponent do contêiner de nome *container_name*.
  - *getContainers*(): Fornece descrições de todos os contêineres ativos em um determinado Nó de Execução. Estas descrições são compostas pela referência remota para a faceta IComponent do contêiner, nome do contêiner e referência remota para a faceta IComponent do Nó de Execução.
  - *getName*(): Fornece o nome da máquina na qual o Nó de Execução reside.

- Faceta *ContainerManager*
  - *registerContainer*(*in string name*, *in core::IComponent ctr*, *in string pid*): Efetua o registro de um contêiner. São enviados o nome, a referência remota e o identificador de processo no sistema operacional(PID) do mesmo. Neste método, o Nó de Execução persiste a

referência para o contêiner no arquivo apropriado, para que possa ser recuperado após uma reinicialização do nó.
- *unregisterContainer*(*in string name*): Desfaz o registro de um contêiner, através de seu nome.

- Faceta *ComponentProperties*

  - *setProperty*(*in execution_node::Property property*): Atribui a propriedade recebida ao conjunto de propriedades do componente, caso a propriedade ainda não exista lá. Caso contrário, apenas atribui o novo valor da propriedade e se deixa de ser ou passa a ser apenas leitura. Caso a propriedade já seja apenas leitura, não é possível alterá-la remotamente.
  - *getProperty*(*in string name*): Retorna a propriedade que atende pelo nome recebido.
  - *getProperties*(): Retorna todas as propriedades do componente.

B.3
Interfaces do Contêiner de Componentes

- Faceta *IComponent*

  - *startup*(): A versão deste método, no contêiner, apenas carrega seu arquivo de inicialização(opcional).
  - *shutdown*(): Este método, no contêiner, primeiramente desfaz seu registro no Nó de Execução. Após isto, executa uma chamada *shutdown*() síncrona em todos os seus componentes. A ordem desta chamada é inversa à ordem de criação dos componentes, para que componentes que agem como pré-requisito a outros não parem de funcionar prematuramente. As chamadas são síncronas também por este motivo. Por fim, os arquivos relativos a componentes armazenados no disco rígido são apagados, o ORB é finalizado e o processo do contêiner e componentes é terminado.

- Faceta *ComponentLoader*

  - *load*(*in ComponentId id*, *in StringSeq args*): Carrega o componente especificado pelo argumento *id*, passando *args* para sua fábrica. Todo componente SCS deve fornecer uma fábrica, que será utilizada pelo contêiner para criar uma nova instância. Ao receber um

pedido novo de carga, o contêiner primeiro verifica em sua *cache* interna e, caso não encontre o componente, tenta obtê-lo nos repositórios conhecidos. Caso o componente seja encontrado este é copiado para a cache interna e carregado. Um componente a ser carregado é composto de seu código, sua descrição e opcionalmente seu arquivo de ajuda. É retornado um ComponentHandle, estrutura representante daquela instância de componente.
  - *unload*(*in ComponentHandle handle*): Remove uma instância de componente, realizando uma chamada a seu método *shutdown*() e removendo suas referências.
  - *getInstalledComponents*(): Informa quais tipos de componentes são conhecidos pelo contêiner, ou seja, quais estão em sua *cache* interna. O contêiner ainda pode carregar outros componentes diferentes, desde que estejam disponíveis em algum repositório ao qual ele esteja conectado.

- Faceta *ComponentCollection*

  - *getComponent*(*in ComponentId id*): Retorna as estruturas *handle* de todos os componentes do tipo *id*. Cada estrutura deste tipo representa uma instância deste componente carregada no contêiner.
  - *getComponents*(): Retorna todas as instâncias de todos os componentes carregados naquele momento no contêiner, na forma de todos os *handles*.

- Faceta *ComponentInterception*

  - *loadInterceptor*(*in ComponentId id*, *in StringSeq args*, *in unsigned long position*, *in string type*): Carrega um interceptador no contêiner, que pode ser do tipo cliente ou servidor. Interceptadores também são componentes comuns, portanto o método os carrega normalmente através da faceta ComponentLoader e depois cuida do registro de interceptadores. Estes são colocados em uma fila do tipo apropriado(cliente ou servidor), que permite a instalação de novos interceptadores em qualquer posição. Alguns interceptadores de sistema, no entanto, podem ser obrigatoriamente os primeiros ou últimos, mas isto se dá de forma transparente ao usuário.
  - *unloadInterceptor*(*in ComponentHandle handle*): Remove o interceptador especificado em *handle* da fila de interceptadores, e realiza o desregistro do mesmo no contêiner através da faceta ComponentLoader.

- *changePosition*(*in unsigned long instance_id*, *in unsigned long position*): Modifica a posição do interceptador com id *instance_id*, colocando-o na posição especificada em *position*.
- *getInterceptorPosition*(*in unsigned long instance_id*): Fornece a posição do interceptador com id *instance_id*.
- *getClientInterceptorsOrder*(): Retorna os *instance_id* de todos os interceptadores do tipo cliente, na mesma ordem em que estão na fila naquele momento.
- *getServerInterceptorsOrder*(): Retorna os *instance_id* de todos os interceptadores do tipo servidor, na mesma ordem em que estão na fila naquele momento.

– Faceta *ComponentSuspension*

- *suspend*(): Suspende as comunicações externas de componentes do contêiner, com o intuito de uma pausa temporária e rápida. Todas as chamadas recebidas e feitas ficam aguardando liberação. Isto inclui também os interceptadores, com a exceção de interceptadores de sistema. O contêiner propriamente dito não é afetado.
- *halt*(): Pára completamente as comunicações externas de componentes do contêiner, com o intuito de uma pausa mais demorada. Todas as chamadas recebidas e feitas recebem uma exceção. Isto inclui também os interceptadores, com a exceção de interceptadores de sistema. O contêiner propriamente dito não é afetado.
- *resume*(): Retoma as comunicações externas de todo os componentes do contêiner. Caso o estado anterior seja suspenso, as chamadas retidas serão resumidas.
- *getStatus*(): Informa o estado atual do contêiner, entre normal, suspenso ou parado.

– Faceta *ComponentHelp*

- *getHelpInfo*(*in container::ComponentId id*): Fornece uma *string* com a ajuda do componente.

– Faceta *ComponentProperties*

- *setProperty*(*in execution_node::Property property*): Atribui a propriedade recebida ao conjunto de propriedades do componente, caso a propriedade ainda não exista lá. Caso contrário, apenas atribui o novo valor da propriedade e se deixa de ser ou passa a ser apenas leitura. Caso a propriedade já seja apenas leitura, não é possível alterá-la remotamente.

- *getProperty*(*in string name*): Retorna a propriedade que atende pelo nome recebido.
- *getProperties*(): Retorna todas as propriedades do componente.

B.4
Interfaces do Repositório de Componentes

- Faceta *IComponent*

  - *startup*(): A versão deste método, no repositório, apenas carrega seu arquivo de inicialização(opcional).
  - *shutdown*(): Este método, no repositório, apenas desfaz suas estruturas de dados internas.

- Faceta *ComponentRepository*

  - *install*(*in container::ComponentId id*, *in string entry_point*, *in boolean shared*, *in container::OctetSeq file*, *in string help_info*, *in string extension*): Instala o componente fornecido no repositório. Devem ser fornecidas sua descrição, seu arquivo-fonte e uma *string* de ajuda. A descrição é composta de seu identificador, um ponto de entrada (que pode ser o nome do módulo a ser carregado), um booleano que indica se apenas uma instância do componente deve ser carregada e sua extensão (lua, zip, java, jar, ...). Após instalado, o componente poderá ser obtido por contêineres ou outros repositórios.
  - *uninstall*(*in container::ComponentId id*): Desinstala o componente especificado.
  - *copy*(*in container::ComponentId id*, *in ComponentRepository rep*): Copia o componente especificado para um outro repositório, referenciado por *rep*.
  - *getComponentFile*(*in container::ComponentId id*): Fornece a sequência de octetos do componente.
  - *getComponentDescription*(*in container::ComponentId id*): Fornece a descrição do componente, que é composta pelos dados fornecidos na instalação, com exceção da sequência de octetos e do texto de ajuda.
  - *getInstalledComponents*(): Fornece as descrições de todos os componentes instalados no repositório.

- Faceta *ComponentHelp*
  - *getHelpInfo*(*in container::ComponentId id*): Fornece uma *string* com a ajuda do componente.

Apêndice C

Códigos da Avaliação Experimental

C.1

Código de Instanciação Sem Uso da Infra-Estrutura

Listing C.1: Código de Instanciação Sem Uso da Infra-Estrutura

```
1  #!/bin/bash
2
3  MYPATH=`pwd`
4  LUACMD="luajit"
5
6  function usage(){
7          echo "Usage: $0 <samples> <volume> <#consumers> <#
               suppliers> <output_dir>"
8          echo "        <samples> : iterations"
9          echo "        <volume>  : how many events each supplier
               sends"
10         echo "        <#consumers> : number of consumers"
11         echo "        <#suppliers> : number of suppliers"
12         echo "        <output_dir> : prefix to output
               directories"
13         echo "        <channeld_host> : IP that is running the
               channeld process"
14         echo "        [-monitor]    : hidden internal last arg to
                monitor all execution"
15         exit 0
16 }
17
18 # Se contiver poucos argumentos, imprimir forma de utilização e
       sair
19 if ((($#<5)); then usage; fi;
20
21 SAMPLES=${1:-10}
```

```
22 VOLUME=${2:-5000}
23 CONSUMERS=${3:-10}
24 SUPPLIERS=${4:-1}
25 LABEL=${5:-'basic'}
26 CHANNELD_HOST=${6:-'127.0.0.1'}
27
28 function monitor(){
29   while true
30   do
31     for file in $( ls ../../execution_node/*.pid 2>/dev/null )
32     do
33       outfile=$(echo $file |sed -e 's%.*/%%' -e 's%\.pid%-
             memory%')
34       outfile="/tmp/$outfile"
35       k=1
36       for pid in $(cat $file)
37       do
38         memory=$(grep Vm /proc/$pid/status 2>/dev/null |awk '{
               print $2}' | tr '\n' ' ')
39         [ -n "$memory" ] && echo $memory >> ${outfile}${k}.dat
40         k=$((k+1))
41       done
42     done
43     sleep 1
44   done
45 }
46
47 function clean()
48 {
49   HOSTS=$1
50   for host in $HOSTS
51   do
52     ssh $host "rm -rf /tmp/*.dat /tmp/*.log /tmp/execution_node
           .ior \
53                $MYPATH/../../container/*1 \
54                $MYPATH/../../execution_node/*.pid"
55   done
56 }
57
58 function make_sample()
59 {
60   # limpa diretórios remotos
61   clean "$CONSUMER_HOSTS $SUPPLIER_HOSTS localhost"
62
63   # execução da base do SCS (execution_node) + IRD + NSD
```

```
64    $( which scs ) || ( echo ERROR: missing 'scs' command in your
          PATH && exit 1 )
65
66    # configuração dos componentes
67    $LUACMD channelconfig.lua $SUPPLIERS $VOLUME
68
69    # contagem de hosts remotos
70    count_consumers=$(echo $CONSUMER_HOSTS|wc -w)
71    count_suppliers=$(echo $SUPPLIER_HOSTS|wc -w)
72
73    if [ $(( SUPPLIERS % count_suppliers)) != 0 ]; then
74      echo "[$0] ERROR: SUPPLIERS must be divisible by #
          SUPPLIERS_HOSTS"
75      exit 1
76    elif [ $(( CONSUMERS % count_consumers)) != 0 ]; then
77      echo "[$0] ERROR: CONSUMERS must be divisible by #
          CONSUMERS_HOSTS"
78      exit 1
79    fi
80
81    QTY=$((CONSUMERS/count_consumers))
82    # inicialização de consumidores
83    for each in $CONSUMER_HOSTS
84    do
85      ssh -n $each "bash -c 'source ~/users/amadeu/.bashrc && cd
          $MYPATH && $LUACMD consumer-mult-config.lua $QTY $((
          SUPPLIERS*VOLUME)) \"-alone\" $QTY $CHANNELD_HOST'" 1>
          logconsumer.$each.txt 2>&1 &
86    done
87
88    echo "waiting start of the consumers"
89    sleep 45
90    echo "take care! suppliers are comming!"
91
92    QTY=$((SUPPLIERS/count_suppliers))
93    # inicialização de produtores
94    for each in $SUPPLIER_HOSTS
95    do
96      ssh -n $each "bash -c 'source ~/users/amadeu/.bashrc && cd
          $MYPATH && $LUACMD supplier-mult-config.lua $QTY
          $VOLUME \"-alone\" $QTY $CHANNELD_HOST'" 1> logsupplier
          .$each.txt 2>&1 &
97    done
98
99    # obtenção de PIDs dos nós de execução
```

```
100     en_pids=$( ps -opid,command -C $LUACMD --no-headers|grep
            Execution|awk '{print $1}' )
101
102     # espera pela finalização dos nós de execução
103     count=0
104     max=$( echo ${en_pids} |wc -w )
105     while ((count < max)); do
106       count=0
107       for pid in $(echo ${en_pids}); do
108         [ ! -d /proc/$pid ] && count=$((count+1))
109       done
110       sleep 2
111     done
112
113     echo "[done]"
114   }
115
116   # código main
117   for ((j=1;j<=$SAMPLES;j++))
118   do
119     mkdir ${LABEL}
120     echo "making sample..."
121
122     if [ "$7" == "-monitor" ]
123     then
124       monitor
125       exit 0
126     fi
127
128     # roda novamente com flag monitor ligada
129     nice -n 19 $0 $@ -monitor &
130     mon_pid="$!"
131
132     make_sample
133
134     # mata todos os processos lua e de monitoração
135     filter_cmd="ps -opid,command -C $LUACMD --no-headers|egrep \"
            ExecutionNode|Container\"|awk '{ print \$1 }'|xargs kill
            -9"
136     echo "killing local lua processes"
137
138     local_pids=$( bash -c "${filter_cmd}" )
139     kill -9 $mon_pid $local_pids
140
141     echo "killing remote lua processes"
```

```
142     returns=""
143     if [ "$CHANNELD_HOST" != "" -a "$CHANNELD_HOST" !=
            "127.0.0.1" ]
144     then
145       for each in $CONSUMER_HOSTS $SUPPLIER_HOSTS
146       do
147         ssh $each "${filter_cmd}"
148         returns="$returns $?"
149       done
150       echo -e "DEBUG: `date +%Y%m%d%t%T` \t $returns"
151     fi
152
153     echo "getting temporary file with benchmark data..."
154     # recuperação dos dados não tratados
155     for host in $CONSUMER_HOSTS $SUPPLIER_HOSTS
156     do
157       ssh $host "rename consumer consumer.$host. /tmp/consumer*.
              dat" 2>/dev/null
158       ssh $host "rename supplier supplier.$host. /tmp/supplier*.
              dat" 2>/dev/null
159       scp $host:/tmp/*.dat /tmp
160     done
161
162     mv log* ${LABEL}/
163     mv /tmp/*.log ${LABEL}/
164
165     mv /tmp/*memory1.dat ${LABEL}/
166     # renomeação baseada no pacote util-linux!
167     rename memory1.dat memory_${j}.dat ${LABEL}/*memory1.dat
168
169     cat /tmp/consumer*.dat > ${LABEL}/consumer_${j}.dat
170     cat /tmp/supplier*.dat > ${LABEL}/supplier_${j}.dat
171     # renomeação baseada no pacote util-linux!
172     rename .log _${j} ${LABEL}/*.log
173     rm ../../execution_node/*.pid
174
175   done
```

C.2

Código de Tratamento dos Resultados Sem Uso da Infra-Estrutura

Listing C.2: Código de Tratamento dos Resultados

```
#!/bin/bash

SAMPLES=${1:-5}
EVENTS=${2:-50000}
SUPPLIERS=1
CONSUMERS=1

function run_octave(){
   file=$1
  # results
   echo "
load '$file/total.txt'
f = fopen('$file/results.txt','w')

fprintf(f,'#media\t\t desvio\t\t max\t\t min\n')
fprintf(f,'%.9f\t %.9f\t %.9f\t %.9f\n',mean(total),std(total),
    max(total),min(total))
fclose(f)
   " |octave -q >/dev/null
   echo "
mem1 = load '$file/ChannelContainer-memory_1.dat'
mem2 = load '$file/ChannelContainer-memory_2.dat'
mem3 = load '$file/ChannelContainer-memory_3.dat'
mem4 = load '$file/ChannelContainer-memory_4.dat'
mem5 = load '$file/ChannelContainer-memory_5.dat'
mem6 = load '$file/ChannelContainer-memory_6.dat'
mem7 = load '$file/ChannelContainer-memory_7.dat'
mem8 = load '$file/ChannelContainer-memory_8.dat'
mem9 = load '$file/ChannelContainer-memory_9.dat'
mem10 = load '$file/ChannelContainer-memory_10.dat'

mem_res = [max(mem1)-min(mem1);max(mem2)-min(mem2);max(mem3)-
    min(mem3);max(mem4)-min(mem4);max(mem5)-min(mem5);max(mem6)
    -min(mem6);max(mem7)-min(mem7);max(mem8)-min(mem8);max(mem9
    )-min(mem9);max(mem10)-min(mem10)]
media = mean(mem_res)
desvio = std(mem_res)

save $file/results-memory.txt media desvio
   " |octave -q >/dev/null
```

```
37 }
38
39 # diretório dos gráficos e logs finais
40 mkdir log-oil 2>/dev/null
41 # limpar diretório temporário
42 rm -f /tmp/{total,chanmem}_[0-9]*_supp.txt
43
44 # Código main
45 for each in $(ls -d report-c* 2>/dev/null); do
46   # total
47   rm -f $each/total.txt
48   # getting number of suppliers from diretory name
49   CONSUMERS=$(echo $each | sed -e 's%.*-c\(.*\)s\(.*\)%\1%')
50   SUPPLIERS=$(echo $each | sed -e 's%.*-c\(.*\)s\(.*\)%\2%')
51   endtime=''
52   starttime=''
53   for ((i=1;i<=$SAMPLES;i++)); do
54     endtime=$(cat $each/dispatcher_$i)
55     starttime=$(cat $each/proxy_$i)
56     lua -e "print(${EVENTS}*${SUPPLIERS}*${CONSUMERS}/(${
         endtime}-${starttime}))" >> $each/total.txt;
57   done;
58   run_octave "$each"
59   echo -e "$CONSUMERS\t $(cat $each/results.txt|grep -v media)"
       >> /tmp/total_${SUPPLIERS}_supp.txt
60   echo -e "$CONSUMERS\t $(cat $each/results-memory.txt |grep -v
       '^#')" >> /tmp/chanmem_${SUPPLIERS}_supp.txt
61   sort -n /tmp/total_${SUPPLIERS}_supp.txt > log-oil/total_${
       SUPPLIERS}_supp.txt
62   sort -n /tmp/chanmem_${SUPPLIERS}_supp.txt > log-oil/
       chanmem_${SUPPLIERS}_supp.txt
63 done;
64
65 # limpar diretório temporário
66 rm -f /tmp/{total,chanmem}_[0-9]*_supp.txt
```