7
Referências Bibliográficas

G. C. N. P. Michael Clarke, Gordon S. Blair, *The OpenCOMJ Handbook*. Lancaster University, 2007. (document), 2, 2.2

B. Meyer, "What to compose," *In the Beyond Objects Column of Software Development Magazine*, Mar. 2000. 1

C. Szyperski, *Component Software: Beyond Object-Oriented Programming*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. 1

I. Crnkovic, S. Sentilles, A. Vulgarakis, and M. R. Chaudron, "A classification framework for software component models," *IEEE Transactions on Software Engineering*, vol. 99, no. PrePrints, 2010. 1, 2, 2.4, 6

E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani, "The FRACTAL component model and its support in java: Experiences with auto-adaptive and reconfigurable systems," *Software: Practice and Experience*, vol. 36, no. 11-12, pp. 1257–1284, 2006. 1, 1, 2

G. Coulson, G. Blair, P. Grace, F. Taiani, A. Joolia, K. Lee, J. Ueyama, and T. Sivaharan, "A generic component model for building systems software," *ACM Transactions on Computer Systems*, vol. 26, no. 1, pp. 1–42, 2008. 1, 1, 2

K.-K. Lau, L. Ling, and P. V. Elizondo, "Towards composing software components in both design and deployment phases," in *Proceedings of the 10th international conference on Component-based software engineering*, ser. CBSE'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 274–282. [Online]. Available: http://portal.acm.org/citation.cfm?id=1770657.1770680 1, 3.2

R. van Ommering, F. van der Linden, J. Kramer, and J. Magee, "The Koala Component Model for Consumer Electronics Software," *Computer*, vol. 33, no. 3, pp. 78–85, 2000. [Online]. Available: http://dx.doi.org/http://dx.doi.org/10.1109/2.825699 1, 2

C. Atkinson, J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wust, and J. Zettel, *Component-Based Product Line Engineering with UML*, 1st ed. Addison-Wesley Professional, Nov. 2001. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0201737914 1, 2

H. Hansson, M. Akerholm, I. Crnkovic, and M. Torngren, "Saveccm - a component model for safety-critical real-time systems," in *EUROMICRO*, 2004, pp. 627–635. 1, 2

C. E. L. Augusto, R. G. Cerqueira, S. Correa, E. Fonseca, L. Marques, and H. Hoenick, "SCS: Software Component System," 2009, http://www.tecgraf.puc-rio.br/~scorrea/scs. 1.1, 3, 4.1

F. A. Portella, R. G. Cerqueira, and Correa, "Um serviço de captura e acesso para espaços ativos," 2008, http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0511020_08_pretextual.pdf. 1.1, 5, 5.1

K.-K. Lau, M. Ornaghi, and Z. Wang, "A software component model and its preliminary formalisation," in *Proc. 4th International Symposium on Formal Methods for Components and Objects, LNCS 4111*, F. de Boer *et al.*, Ed. Springer-Verlag, 2006, pp. 1–21. 2.3.4

N. Russell, A. H. M. T. Hofstede, and N. Mulyar, "Workflow controlflow patterns: A revised view," Tech. Rep., 2006. 2.3.4

E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software.* Reading, MA: Addison-Wesley, 1995. 2.3.4

P.-C. David, T. Ledoux, M. Léger, and T. Coupaye, "Fpath and fscript: Language support for navigation and reliable reconfiguration of fractal architectures," *Annals of Telecommunications*, vol. 64, pp. 45–63, 2009, 10.1007/s12243-008-0073-y. [Online]. Available: http://dx.doi.org/10.1007/s12243-008-0073-y 2.3.8

T. V. Batista, A. Joolia, and G. Coulson, "Managing dynamic reconfiguration in component-based systems," in *Proceedings of the 2nd European Workshop Software Architecture (EWSA'05)*, ser. Lecture Notes in Computer Science, vol. 3527. Springer, 2005, pp. 1–17. 2.3.8

D. Box, *Essential COM (DevelopMentor Series).* Addison-Wesley Professional, Jan. 1998. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0201634465 3.1

OMG, "CORBA Component Model specification," Object Management Group, Tech. Rep., 2006, http://www.omg.org/technology/documents/formal/components.htm. 3.1

B. Liskov, "Keynote address - data abstraction and hierarchy," in *Addendum to the proceedings on Object-oriented programming systems, languages and applications (Addendum)*, ser. OOPSLA '87. New York, NY, USA: ACM, 1987, pp. 17–34. [Online]. Available: http://doi.acm.org/10.1145/62138.62141 3.1.2

B. Stroustrup, *The C++ Programming Language, Third Edition*. Addison-Wesley, 2000. 4

J. Gosling, B. Joy, G. Steele, and G. Bracha, *Java (TM) Language Specification, The (Java (Addison-Wesley))*. Addison-Wesley Professional, 2005. 4

E. International, *Standard ECMA-334 - C# Language Specification*, 4th ed., June 2006. [Online]. Available: http://www.ecma-international.org/publications/standards/Ecma-334.htm 4

R. Ierusalimschy, L. H. de Figueiredo, and W. Celes, *Lua 5.1 Reference Manual*. Lua.org, 2006. [Online]. Available: http://www.amazon.com/exec/obidos/ASIN/8590379833/lua-indexmanual-20 4

Tecgraf/PUC-Rio, "Openbus: Um middleware para integração de aplicações baseadas em componentes," http://www.tecgraf.puc-rio.br/openbus. 5.1

H. M. C. Saldanha, "Utilizando anotações em linguagens orientadas a objetos para suporte à programação orientada a componentes," Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ, Brasil, Aug. 2010. 6

A
Middleware SCS 2.0

```
1 package scs2;
2
3 public interface IComponent {
4   final static String interface_id = "component";
5   IFacet getFacet(String name);
6   IFacet[] getFacets(Class cl);
7   IFacet[] getFacets();
8   IFacet[] getFacets(String[] facet_names);
9 }
```

Código A.1: Interface que representa o tipo "'componente"' no SCS.

```
1 package scs2;
2
3 import scs2.exceptions.*;
4
5 public interface IReceptacles {
6   final static String interface_id = "receptacles";
7     IReceptacle getReceptacle(String recep_name);
8   IReceptacle[] getReceptacles();
9   IReceptacle[] getReceptacles(String[] recep_names);
10  void addConnectionListener(IConnectionListener listener);
11  void addConnectionListener(String recep_name,
        IConnectionListener listener)
12       throws NonExistentReceptacleException;
13  void removeListener  (IConnectionListener listener);
14 }
```

Código A.2: Interface que gerência as conexões realizadas por um componente.

```
1 package scs2;
2
3 public interface IFacet {
4   FacetDescription getDescription();
5   Object getObject();
6   IComponent getComponent();
7 }
```

Código A.3: Interface que representa a abstração de uma faceta.

```
1 package scs2;
2
3 import scs2.exceptions.*;
4
5 public interface IReceptacle {
6   ReceptacleDescription getDescription();
7     IFacet[] getConnections();
8     IComponent getComponent();
9   void addConnection(IFacet fct)
10       throws InvalidInterfaceException, AlreadyConnectedException,
11              ExceededConnectionLimitException,
                   IllegalBindingException ;
12  void removeConnection(IFacet fct)
13       throws NoConnectionException;
14 }
```

Código A.4: Interface que representa a abstração de receptáculo.

```
1 package scs2;
2
3 import scs2.exceptions.*;
4
5 public interface ILifeCycle {
6   final static String interface_id = "life_cycle";
7   IReceptacle[] getMissingDependences();
8     void resolveDependences() throws FeatureNotSupportedException;
9   void startup() throws StartupFailedException;
10  void suspend() throws SuspendFailedException;
11  void shutdown() throws ShutdownFailedException;
12 }
```

Código A.5: Interface para manipular o ciclo de vida de um componente.

```
1 package scs2;
2 import scs2.exceptions.*;
3
4 public interface IComponentContext {
5     final static String interface_id = "component_context";
6   IComponent lookup(String name);
7   void bind(String name, IComponent component) throws
       AlreadyBoundException;
8   IComponent unbind(String name) throws CannotUnbindException;
9   String[] getBindings();
10 }
```

Código A.6: Interface que representa o contexto de um aplicação.

```
1 package scs2;
```

```
2
3 import scs2.exceptions.*;
4
5 public interface IComponentAssembler {
6   final static String interface_id = "component_assembler";
7   void startComponent() throws ComponentUnderconstructionException
        ;
8   IComponent finishComponent() throws
        NoComponentUnderconstructionException;
9   void addFacet(FacetDescription descr, Object facet) throws
        NoComponentUnderconstructionException,
        FacetAlreadyExistsException, InvalidInterfaceException;
10  void addReceptacle(ReceptacleDescription descr) throws
        NoComponentUnderconstructionException,
        ReceptacleAlreadyExistsException;
11 }
```

Código A.7: Interface que disponibiliza mecanismos para criação de componentes

B
Exemplos de Uso

B.1
Mapeamento de Facetas

```
1 package component.hello;
2
3 import scs2.*;
4
5 public class A {
6
7     public static IComponent getInstance()
8     {
9         IComponent self = null;
10        final A tst;
11        tst = new A();
12
13        IComponentContext cnx = SCS.getInstance();
14        IComponent cmp = cnx.lookup("component_manager");
15        IFacet fct = cmp.getFacet(IComponentAssemblerFactory.
              interface_id);
16
17        IComponentAssembler assembler;
18        assembler = ((IComponentAssemblerFactory) fct.getObject())
              .create("composite");
19
20        try {
21            assembler.startComponent();
22            self = assembler.finishComponent();
23        } catch (Exception ex) {
24            System.out.println(ex);
25        }
26        return self;
27    }
28
29 }
```

Código B.1: Componente A.

```
1 package component.hello;
```

```
2
3 import scs2.*;
4
5 public class B implements IPrint {
6
7     private IHello hello;
8
9     public static IComponent getInstance()
10    {
11
12        IComponent self = null;
13        final B tst;
14        tst = new B();
15
16        IComponentContext cnx = SCS.getInstance();
17        IComponent cmp = cnx.lookup("component_manager");
18        IFacet fct = cmp.getFacet(IComponentAssemblerFactory.
              interface_id);
19
20        IComponentAssembler assembler;
21        assembler = ((IComponentAssemblerFactory) fct.getObject())
              .create();
22
23        try {
24            assembler.startComponent();
25            assembler.addFacet(new FacetDescription("helloFctB",
                  IPrint.class), tst);
26            assembler.addReceptacle(new ReceptacleDescription("
                  helloRecp", IHello.class, 0, false));
27            self = assembler.finishComponent();
28
29            IReceptacles recps = (IReceptacles) self.getFacet(
                  IReceptacles.interface_id).getObject();
30
31            recps.addConnectionListener(new IConnectionListener()
                  {
32
33                public void notifyConnection(IReceptacle rcp,
                      IFacet fct) {
34
35                    Object obj = fct.getObject();
36                    if (obj instanceof IHello) {
37                        tst.setHello((IHello)obj);
38                    }
39                }
40
41                public void notifyDisconnection(IReceptacle rcp,
```

```
                    IFacet fct) {
42
43                      Object obj = fct.getObject();
44
45                      if (obj instanceof C) {
46                          tst.setHello(null);
47                      }
48                  }
49              });
50
51
52          } catch (Exception ex) {
53              System.out.println(ex);
54          }
55          return self;
56      }
57
58      public void printHello()
59      {
60          hello.sayHello();
61      }
62
63      public void setHello(IHello hello)
64      {
65          this.hello = hello;
66      }
67
68 }
```

Código B.2: Componente B.

```
1 package component.hello;
2
3 import scs2.*;
4
5 public class C implements IHello {
6
7     public static IComponent getInstance() {
8         IComponent self = null;
9         final C tst;
10        tst = new C();
11
12        IComponentContext cnx = SCS.getInstance();
13        IComponent cmp = cnx.lookup("component_manager");
14        IFacet fct = cmp.getFacet(IComponentAssemblerFactory.
              interface_id);
15
16        IComponentAssembler assembler;
```

```
17          assembler = ((IComponentAssemblerFactory) fct.getObject())
                .create();
18
19          try {
20              assembler.startComponent();
21              assembler.addFacet(new FacetDescription("helloFctC",
                    IHello.class), tst);
22              self = assembler.finishComponent();
23
24          } catch (Exception ex) {
25              System.out.println(ex);
26          }
27          return self;
28      }
29
30      public void sayHello() {
31         System.out.println("Hello! I'm component C.");
32      }
33 }
```

Código B.3: Componente C.

```
 1 package component.hello;
 2
 3 import scs2.*;
 4
 5 public class RunHello {
 6
 7      public static void main(String [] args)
 8      {
 9          SCSContext cnx = null;
10
11          try {
12
13              cnx = (SCSContext) SCS.getInstance();
14
15              IComponent cmpA = (IComponent) A.getInstance();
16              IComponent cmpB = (IComponent) B.getInstance();
17              IComponent cmpC = (IComponent) C.getInstance();
18
19              IContentController compositeService = (
                    IContentController) cmpA.getFacet(
                    IContentController.interface_id).getObject();
20
21              compositeService.addSubComponent(cmpB);
22              compositeService.addSubComponent(cmpC);
23
24              IFacet fct = cmpC.getFacet("helloFctC");
```

```
25
26              IReceptacles recps = (IReceptacles)cmpB.getFacet(
                    IReceptacles.interface_id).getObject();
27              IReceptacle recp = recps.getReceptacle("helloRecp");
28              recp.addConnection(fct);
29
30              compositeService.bindFacet(0,"helloFctB","
                    externalHello");
31
32              IFacet helloFacet = cmpA.getFacet("externalHello");
33              IPrint helloImpl = (IPrint) helloFacet.getObject();
34              helloImpl.printHello();
35
36          }
37          catch(Exception e)
38          {
39
40          }
41      }
42
43 }
```

Código B.4: Classe *Main* para executar o exemplo SCS local.

B.2 Mapeamento de Receptáculos

```
 1 local oo = require "loop.base"
 2 local oil = require "oil"
 3 local orb = oil.init({host="localhost",port=1040})
 4 oil.orb = orb
 5
 6 local primitive = require "scs.core.base"
 7 local scs = require "scs.core.composite"
 8
 9 orb:loadidlfile("../idl/scs.idl")
10 orb:loadidlfile("../idl/membrane.idl")
11
12 -- criação do ComponentId
13 local cpId = {
14   name = "Composite Component",
15   major_version = 1,
16   minor_version = 0,
17   patch_version = 0,
18   platform_spec = ""
19 }
20
```

```
21
22 oil.verbose:level(0)
23 oil.main(function ()
24
25   oil.newthread(orb.run,orb)
26   instance = scs.newCompositeComponent({},{},cpId)
27   oil.writeto("componentA.ior",orb:tostring(instance.IComponent))
28
29 end)
```

Código B.5: Componente A.

```
 1 local oo = require "loop.base"
 2 local oil = require "oil"
 3 local orb = oil.init({host="localhost",port=1065})
 4 oil.orb = orb
 5
 6 local scs = require "scs.core.base"
 7
 8 oil.verbose:level(0)
 9 orb:loadidlfile("../idl/scs.idl")
10 orb:loadidlfile("pingPong.idl")
11
12
13 local facetDescs = {}
14 facetDescs.PingPongServer = {
15   name = "PingPongServer",
16   interface_name = "IDL:scs/demos/pingpong/PingPongServer:1.0",
17   class = PingPongServer
18 }
19
20 local receptDescs = {}
21 receptDescs.PingPongReceptacle  = {}
22 receptDescs.PingPongReceptacle.name    = "PingPongReceptacle"
23 receptDescs.PingPongReceptacle.interface_name = "IDL:scs/demos/
       pingpong/PingPongServer:1.0"
24 receptDescs.PingPongReceptacle.is_multiplex = false
25 receptDescs.PingPongReceptacle.type    = "Receptacle"
26
27 componentId = { name = "PingPong", major_version = 1,
       minor_version = 0,
28   patch_version = 0, platform_spec = "" }
29
30
31 oil.main(function ()
32
33   oil.newthread(orb.run,orb)
```

```
34    basicComponent = scs.newComponent(facetDescs,receptDescs,
          componentId)
35    oil.writeto("componentB.ior", orb:tostring(basicComponent.
          IComponent))
36
37 end)
```

Código B.6: Componente B.

```
 1 local oo = require "loop.base"
 2 local oil = require "oil"
 3 local orb = oil.init({host="localhost",port=1040})
 4 oil.orb = orb
 5
 6 local scs = require "scs.core.base"
 7
 8 oil.verbose:level(0)
 9 orb:loadidlfile("../idl/scs.idl")
10 orb:loadidlfile("pingPong.idl")
11
12
13 local facetDescs = {}
14 facetDescs.PingPongServer = {
15   name = "PingPongServer",
16   interface_name = "IDL:scs/demos/pingpong/PingPongServer:1.0",
17   class = PingPongServer
18 }
19
20 local receptDescs = {}
21 receptDescs.PingPongReceptacle  = {}
22 receptDescs.PingPongReceptacle.name    = "PingPongReceptacle"
23 receptDescs.PingPongReceptacle.interface_name = "IDL:scs/demos/
       pingpong/PingPongServer:1.0"
24 receptDescs.PingPongReceptacle.is_multiplex = false
25 receptDescs.PingPongReceptacle.type    = "Receptacle"
26
27 componentId = { name = "PingPong", major_version = 1,
       minor_version = 0,
28   patch_version = 0, platform_spec = "" }
29
30
31 oil.main(function ()
32
33   oil.newthread(orb.run,orb)
34   basicComponent = scs.newComponent(facetDescs,receptDescs,
         componentId)
35   oil.writeto("componentC.ior", orb:tostring(basicComponent.
         IComponent))
```

```
36
37 end)
```

Código B.7: Componente C

```
1 local PingPongServer = oo.class{ id = 0, stop = false }
2
3 function PingPongServer:ping()
4   if self.stop == true then
5     return
6   end
7   local otherPP = self.context.PingPongReceptacle
8   print("PingPong " .. self.id .. " received ping from PingPong "
        .. otherPP:getId() .. "! Ponging in 3 seconds...")
9   oil.sleep(3)
10  oil.newthread(otherPP.pong, otherPP)
11 end
12
13 function PingPongServer:pong()
14  if self.stop == true then
15    return
16  end
17  local otherPP = self.context.PingPongReceptacle
18  print("PingPong " .. self.id .. " received pong from PingPong "
        .. otherPP:getId() .. "! Pinging in 3 seconds...")
19  oil.sleep(3)
20  oil.newthread(otherPP.ping, otherPP)
21 end
22
23 function PingPongServer:setId(id)
24  self.id = id
25 end
26
27 function PingPongServer:getId()
28  return self.id
29 end
30
31 function PingPongServer:start()
32  print("PingPong " .. self.id .. " received an start call!")
33  self.stop = false
34  local otherPP = self.context.PingPongReceptacle
35  oil.newthread(otherPP.ping, otherPP)
36 end
37
38 function PingPongServer:stop()
39  self.stop = true
40 end
```

Código B.8: Classe Ping-Pong.

```
1 local oil = require "oil"
2 local orb = oil.init()
3
4 orb:loadidlfile("../idl/scs.idl")
5 orb:loadidlfile("../idl/composite.idl")
6 orb:loadidlfile("pingPong.idl")
7 oil.verbose:level(0)
8 oil.main(function()
9
10   local primitiveComponentIOR = oil.readfrom("basic_component.ior
       ")
11   local primitiveComponent = orb:newproxy(primitiveComponentIOR)
12
13   local contentControllerIOR = oil.readfrom("content_controller.
       ior")
14   local contentControllerComponent = orb:newproxy(
       contentControllerIOR)
15   contentControllerComponent = orb:narrow(
       contentControllerComponent,"IDL:scs/core/IComponent:1.0")
16
17   local externalComponentIOR = oil.readfrom("basic_component1.ior
       ")
18   local externalComponent = orb:newproxy(externalComponentIOR)
19
20   primitiveComponent:startup()
21   contentControllerComponent:startup()
22   externalComponent:startup()
23
24   if primitiveComponent then
25
26     local compositeFacet  = contentControllerComponent:
         getFacetByName("IContentController")
27     compositeFacet = orb:narrow(compositeFacet,"IDL:scs/core/
         IContentController:1.0")
28
29     compositeFacet:addSubComponent(primitiveComponent)
30
31     compositeFacet:bindFacet(0,"PingPongServer","PingPongServer")
32     compositeFacet:bindReceptacle(0,"PingPongReceptacle","
         PingPongReceptacle")
33
34     local receptaclesFacet = contentControllerComponent:
         getFacetByName("IReceptacles")
35     receptaclesFacet = orb:narrow(receptaclesFacet,"IDL:scs/core/
         IReceptacles:1.0")
```

```
36
37      local externalRecp = externalComponent:getFacetByName("
            IReceptacles")
38      externalRecp = orb:narrow(externalRecp,"IDL:scs/core/
            IReceptacles:1.0")
39
40      compositePingPongServer = contentControllerComponent:
            getFacetByName("PingPongServer")
41      compositePingPongServer = orb:narrow(compositePingPongServer,"
            IDL:scs/demos/pingpong/PingPongServer:1.0")
42
43      externalPingPongServer = externalComponent:getFacetByName("
            PingPongServer")
44      externalPingPongServer  = orb:narrow(externalPingPongServer,"
            IDL:scs/demos/pingpong/PingPongServer:1.0")
45
46      receptaclesFacet:connect("PingPongReceptacle",
            externalPingPongServer)
47      externalRecp:connect("PingPongReceptacle",
            compositePingPongServer)
48
49      compositePingPongServer:start()
50      print("Pings and Pongs started!")
51
52    end
53
54 end)
```

Código B.9: Script para execução do Ping-Pong.

C
Interfaces do CAS

```
1 #ifndef ROOM
2 #define ROOM
3
4 #include "monitoring.idl"
5
6 module cas {
7
8   module room {
9
10    interface IRoom {
11
12    string getStatus();
13    string getName();
14
15    string getCurrentEventProfile();
16    void setCurrentEventProfile(in string profile);
17    long getCurrentEventID();
18    void setCurrentEventID(in long eventID);
19    long addFailMonitor(in cas::monitoring::IFailEvent failMonitor
          );
20    };
21  };
22 };
23
24 #endif
```

Código C.1: Interface *IRoom* que oferece operações **set/get** sobre propriedades de um ambiente.

```
1 #ifndef RECORDER
2 #define RECORDER
3
4 #include "dataRepository.idl"
5
6 module cas {
7
8    module recorder {
9   exception NotRecording { string msg; };
```

```
10   exception AlreadyRecording { string msg; };
11   exception NotConfigured { string msg; };
12   exception StartRecordFailure { string reason; };
13   interface IRecord {
14     void startRecord() raises(NotConfigured, AlreadyRecording,
         StartRecordFailure);
15     string getStatus();
16     void stopRecord() raises(NotRecording);
17     void discardRecordedData() raises(NotRecording);
18     };
19   };
20 };
21
22 #endif
```

Código C.2: Interface *IRecord* que disponibiliza funcionalidades genéricas para um dispositivo que captura mídia. Possui métodos para iniciar e parar uma gravação e retornar o status do dispositivo.


```
1 #ifndef DATA_IDL
2 #define DATA_IDL
3
4 #include "dataRepository.idl"
5
6 module cas {
7
8    module transfer {
9
10     exception DataNotAvailable { string msg; };
11     exception DataTransferError { string msg; };
12
13     interface IProgressListener {
14       void notifyProgress(in cas::data::Event eventDescriptor, in
           float progress);
15       void finished(in cas::data::Event eventDescriptor);
16       void error(in cas::data::Event eventDescriptor, in string
           reason);
17     };
18
19     interface IDataTransfer {
20       void transferRecordedData() raises(DataNotAvailable,
           DataTransferError);
21       void transferRecordedDataFromEvent(in cas::data::Event
           eventDescriptor) raises(DataNotAvailable,
           DataTransferError);
22       string getStatus(in cas::data::Event eventDescriptor);
23       void addProgressListener(in IProgressListener listener);
```

```
24        void removeProgressListener(in IProgressListener listener);
25        float getProgress();
26      };
27    };
28 };
29
30 #endif
```

Código C.3: Interface *IDataTransfer* que define métodos para transferência de dados após término da gravação de uma mídia.

```
1 #ifndef CONFIGURABLE
2 #define CONFIGURABLE
3
4 module cas {
5
6   module configuration {
7
8     struct Property {
9       string name;
10    };
11
12    exception UnsupportedProperty { string msg; };
13  exception MalformedValue { string msg; };
14  exception CannotSetProperty { string msg; };
15
16  typedef sequence<Property> supportedProperties;
17    typedef sequence<string> supportedPropertiesNames;
18
19    interface IConfigurable {
20
21      void setProperty(in Property prop) raises (
            UnsupportedProperty, CannotSetProperty, MalformedValue);
22      Property getProperty(in string key) raises (
            UnsupportedProperty);
23      supportedProperties getProperties();
24      supportedPropertiesNames getSupportedProperties();
25    };
26  };
27 };
28
29 #endif
```

Código C.4: Interface *IConfigurable* que disponibiliza métodos **set/get** das propriedades específicas do componente que a implementa.

```
1 #ifndef POST_PROCESSOR
2 #define POST_PROCESSOR
```

```
3
4 #include "dataRepository.idl"
5
6 module cas {
7
8   module postProcess {
9
10  interface IPostProcessListener {
11
12      void notifyProgress(in cas::data::Event eventDescriptor, in
            float progress);
13      void finished(in cas::data::Event eventDescriptor);
14      void error(in cas::data::Event eventDescriptor, in string
            reason);
15    };
16
17    interface IPostProcessor {
18    supportedFormats getInputSupportedFormats();
19    supportedFormats getOutputFormats(in string inputFormat);
20    void postProcess(in cas::data::Event eventDescriptor);
21    unsigned long addPostProcessListener(in IPostProcessListener
        listener);
22    void removePostProcessListener(in unsigned long listener);
23    float getPostProcessProgress();
24    string getStatus(in cas::data::Event eventDescriptor);
25    };
26  };
27 };
28
29 #endif
```

Código C.5: Interface *IPostProcessor* que oferece operações de pós-processamento para mídias capturadas.

```
1 module cas {
2   module room {
3     typedef long ConnectionId;
4     interface IRoomConfigurator {
5
6       ConnectionId connectComponent(in Object comp);
7       boolean disconnectComponent(in ConnectionId connection_Id);
8     };
9   };
10 };
```

Código C.6: Interface IRoomConfigurator.