

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 FRAC-TAL 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. Tornngren, “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,
12                                IConnectionListener listener)
13        throws NonExistentReceptacleException;
14    void removeListener (IConnectionListener listener);
15 }
```

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,
12             IllegalBindingException;
13     void removeConnection(IFacet fct)
14         throws NoConnectionException;
15 }

```

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 [] getMissingDependencies();
8     void resolveDependencies() 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
8         AlreadyBoundException;
9     IComponent unbind(String name) throws CannotUnbindException;
10    String [] getBindings();
11 }

```

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         ;
9     IComponent finishComponent() throws
10         NoComponentUnderconstructionException;
11     void addFacet(FacetDescription descr, Object facet) throws
12         NoComponentUnderconstructionException,
13         FacetAlreadyExistsException, InvalidInterfaceException;
14     void addReceptacle(ReceptacleDescription descr) throws
15         NoComponentUnderconstructionException,
16         ReceptacleAlreadyExistsException;
17 }
```

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.
16            interface_id);
17
18        IComponentAssembler assembler;
19        assembler = ((IComponentAssemblerFactory) fct.getObject())
20            .create("composite");
21
22        try {
23            assembler.startComponent();
24            self = assembler.finishComponent();
25        } catch (Exception ex) {
26            System.out.println(ex);
27        }
28
29        return self;
30    }
31 }
```

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.
19            interface_id);
20
21        IComponentAssembler assembler;
22        assembler = ((IComponentAssemblerFactory) fct.getObject())
23            .create();
24
25        try {
26            assembler.startComponent();
27            assembler.addFacet(new FacetDescription("helloFctB",
28                IPrint.class), tst);
29            assembler.addReceptacle(new ReceptacleDescription("
30                helloRecp", IHello.class, 0, false));
31            self = assembler.finishComponent();
32
33            IReceptacles recps = (IReceptacles) self.getFacet(
34                IReceptacles.interface_id).getObject();
35
36            recps.addConnectionListener(new IConnectionListener()
37                {
38
39                    public void notifyConnection(IReceptacle rcp,
40                        IFacet fct) {
41
42                        Object obj = fct.getObject();
43                        if (obj instanceof IHello) {
44                            tst.setHello((IHello)obj);
45                        }
46                    }
47                }
48            );
49
50            public void notifyDisconnection(IReceptacle rcp,
```



```

42         IFacet fct) {
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.
15            interface_id);
16
17        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(
27             IReceptacles.interface_id).getObject();
28         IReceptacle recp = recps.getReceptacle("helloRecp");
29         recp.addConnection(fct);
30
31         compositeService.bindFacet(0,"helloFctB",
32             externalHello");
33
34         IFacet helloFacet = cmpA.getFacet("externalHello");
35         IPrint helloImpl = (IPrint) helloFacet.getObject();
36         helloImpl.printHello();
37     }
38     catch(Exception e)
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 "
9     .. otherPP:getId() .. "! Ponging in 3 seconds...")
10  oil.sleep(3)
11  oil.newthread(otherPP.pong, otherPP)
12 end
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 "
19     .. otherPP:getId() .. "! Pinging in 3 seconds...")
20   oil.sleep(3)
21   oil.newthread(otherPP.ping, otherPP)
22 end
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  ")
12  local primitiveComponent = orb:newproxy(primitiveComponentIOR)
13  local contentControllerIOR = oil.readfrom("content_controller.
14  ior")
15  local contentControllerComponent = orb:newproxy(
16  contentControllerIOR)
17  contentControllerComponent = orb:narrow(
18  contentControllerComponent,"IDL:scs/core/IComponent:1.0")
19
20  local externalComponentIOR = oil.readfrom("basic_component1.ior
21  ")
22  local externalComponent = orb:newproxy(externalComponentIOR)
23
24  primitiveComponent:startup()
25  contentControllerComponent:startup()
26  externalComponent:startup()
27
28  if primitiveComponent then
29
30    local compositeFacet = contentControllerComponent:
31    getFacetByName("IContentController")
32    compositeFacet = orb:narrow(compositeFacet,"IDL:scs/core/
33    IContentController:1.0")
34
35    compositeFacet:addSubComponent(primitiveComponent)
36
37    compositeFacet:bindFacet(0,"PingPongServer","PingPongServer")
38    compositeFacet:bindReceptacle(0,"PingPongReceptacle","
39    PingPongReceptacle")
40
41    local receptaclesFacet = contentControllerComponent:
42    getFacetByName("IReceptacles")
43    receptaclesFacet = orb:narrow(receptaclesFacet,"IDL:scs/core/
44    IReceptacles:1.0")
```

```
36
37     local externalRecp = externalComponent :getFacetByName("
38         IReceptacles")
39
40     compositePingPongServer = contentControllerComponent :
41         getFacetByName(" PingPongServer")
42
43     externalPingPongServer = externalComponent :getFacetByName("
44         PingPongServer")
45
46     externalPingPongServer = orb:narrow(externalPingPongServer,"
47         IDL:scs/demos/pingpong/PingPongServer:1.0")
48
49     receptaclesFacet :connect(" PingPongReceptacle",
50         externalPingPongServer)
51     externalRecp :connect(" PingPongReceptacle",
52         compositePingPongServer)
53
54     compositePingPongServer :start()
55     print(" Pings and Pongs started!")
56
57 end
58 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     };
11 }
12 #endif
```

```

10  exception AlreadyRecording { string msg; };
11  exception NotConfigured { string msg; };
12  exception StartRecordFailure { string reason; };
13  interface IRecord {
14      void startRecord() raises(NotConfigured, AlreadyRecording,
15          StartRecordFailure);
16      string getStatus();
17      void stopRecord() raises(NotRecording);
18      void discardRecordedData() raises(NotRecording);
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
15                 float progress);
16             void finished(in cas::data::Event eventDescriptor);
17             void error(in cas::data::Event eventDescriptor, in string
18                 reason);
19         };
20
21         interface IDataTransfer {
22             void transferRecordedData() raises(DataNotAvailable,
23                 DataTransferError);
24             void transferRecordedDataFromEvent(in cas::data::Event
25                 eventDescriptor) raises(DataNotAvailable,
26                 DataTransferError);
27             string getStatus(in cas::data::Event eventDescriptor);
28             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 (
22             UnsupportedProperty, CannotSetProperty, MalformedValue);
23         Property getProperty(in string key) raises (
24             UnsupportedProperty);
25         supportedProperties getProperties();
26         supportedPropertiesNames getSupportedProperties();
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
13                float progress);
14            void finished(in cas::data::Event eventDescriptor);
15            void error(in cas::data::Event eventDescriptor, in string
16                reason);
17        };
18
19        interface IPostProcessor {
20            supportedFormats getInputSupportedFormats();
21            supportedFormats getOutputFormats(in string inputFormat);
22            void postProcess(in cas::data::Event eventDescriptor);
23            unsigned long addPostProcessListener(in IPostProcessListener
24                listener);
25            void removePostProcessListener(in unsigned long listener);
26            float getPostProcessProgress();
27            string getStatus(in cas::data::Event eventDescriptor);
28        };
29    };
30 };
31 #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*.