

8

Referências Bibliográficas

- [1] GRONE, B.; KNOPFEL, A.; TABELING, P. Component vs. component: Why we need more than one definition. In: **Proceedings of ECBS'05: 12th IEEE International Conference and Workshops on Engineering of Computer-Based Systems**. Washington, DC, USA: IEEE Computer Society, 2005. p. 550–552. ISBN 0-7695-2308-0. 1
- [2] SZYPERSKI, C. **Component Software: Beyond Object-Oriented Programming**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0201745720. 1
- [3] SZYPERSKI, C. Component technology: what, where, and how? In: **Proceedings of the 25th International Conference on Software Engineering (ICSE'03)**. Washington, DC, USA: IEEE Computer Society, 2003. p. 684–693. ISBN 0-7695-1877-X. 1, 2.2.5, 4.1, 7
- [4] EMMERICH, W. Distributed component technologies and their software engineering implications. In: **Proceedings of the 24th International Conference on Software Engineering (ICSE'02)**. New York, NY, USA: ACM, 2002. p. 537–546. ISBN 1-58113-472-X. 1, 2.1, 7
- [5] MANCINELLI, F. et al. Managing the complexity of large free and open source package-based software distributions. In: **Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)**. Washington, DC, USA: IEEE Computer Society, 2006. p. 199–208. ISBN 0-7695-2579-2. 1, 4.1
- [6] FLISSI, A. et al. Deploying on the grid with DeployWare. In: **Proceedings of the 8th International Symposium on Cluster Computing and the Grid (CCGRID'08)**. IEEE Computer Society, 2008. p. 177–184. Disponível em: <<http://dblp.uni-trier.de/db/conf/ccgrid/ccgrid2008.html#FlissiDDM08>>. (document), 1, 1, 2.3.7, 2.13, 5, 5.5
- [7] LACOUR, S.; PÉREZ, C.; PRIOL, T. Deploying CORBA components on a computational grid: General principles and early experiments using

- the Globus Toolkit. In: **Proceedings of the 2nd International Working Conference on Component Deployment (CD'04)**. Springer-Verlag, 2004. (Lecture Notes in Computer Science, 3083), p. 35–49. Disponível em: <<http://www.irisa.fr/paris/Biblio/Papers/Lacour/LacPerPri04CD.pdf>>. 1
- [8] FOSTER, I. T. The anatomy of the grid: Enabling scalable virtual organizations. In: **Euro-Par '01: Proceedings of the 7th International Euro-Par Conference on Parallel Processing**. London, UK: Springer-Verlag, 2001. p. 1–4. ISBN 3-540-42495-4. 1
- [9] SUN MICROSYSTEMS. **Enterprise JavaBeans v3.0**. Santa Clara, USA, maio 2006. Document: JSR/220. 1, 2.3.1, 2.3.1
- [10] OBJECT MANAGEMENT GROUP. **CORBA Components - Version 3.0**. Needham, USA, jun. 2002. Document: formal/2002-06-65. 1, 2.3.3, 2.3.3, 2.3.3, 2.3.4, 2.4, 3.1
- [11] COULSON, G. et al. A generic component model for building systems software. **ACM Transactions on Computer Systems**, ACM, New York, NY, USA, v. 26, n. 1, p. 1–42, 2008. ISSN 0734-2071. (document), 1, 2.3.2, 2.3, 2.3.2
- [12] BRUNETON, E. et al. The FRACTAL component model and its support in java: Experiences with auto-adaptive and reconfigurable systems. **Software: Practice and Experience**, John Wiley & Sons, Inc., New York, NY, USA, v. 36, n. 11-12, p. 1257–1284, 2006. ISSN 0038-0644. 1, 2.3.7, 2.3.7
- [13] DENG, G. et al. DANCE: A QoS-enabled component deployment and configuration engine. In: **Proceedings of the 3rd Working Conference on Component Deployment (CD'05)**. [s.n.], 2005. p. 67–82. Disponível em: <<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.60.36>>. (document), 1, 2.3.4, 2.8, 2.3.4
- [14] OBJECT MANAGEMENT GROUP. **Deployment and Configuration of Component-based Distributed Applications - Version 4.0**. Needham, USA, abr. 2006. Document: formal/2006-04-02. 1, 2.3.4, 2.4
- [15] FLISSI, A.; MERLE, P. A generic deployment framework for grid computing and distributed applications. In: **Proceedings of the 2nd**

- International OTM Symposium on Grid computing, high-performAnce and Distributed Applications (GADA'06)**. [S.l.]: Springer-Verlag, 2006. (Lecture Notes in Computer Science, v. 4279), p. 1402–1411. 1, 2.3.7
- [16] MANOLIOS, P.; VROON, D.; SUBRAMANIAN, G. Automating component-based system assembly. In: **Proceedings of ISSTA'07: International Symposium on Software Testing and Analysis**. New York, NY, USA: ACM, 2007. p. 61–72. ISBN 978-1-59593-734-6. 1
- [17] LACOUR, S.; PEREZ, C.; PRIOL, T. Generic application description model: toward automatic deployment of applications on computational grids. **The 6th IEEE/ACM International Workshop on Grid Computing (CCGRID'05)**, nov. 2005. (document), 1, 2.3.5, 2.9, 2.3.6, 2.4
- [18] CUDENNEC, L. **CoRDAGe : Un service générique de co-déploiement et redéploiement d'applications sur grilles**. Tese (Thèse de doctorat) — Université de Rennes 1, IRISA, Rennes, France, jan. 2009. 1, 7
- [19] MATEVSKA-MEYER, J.; HASSELBRING, W.; REUSSNER, R. H. Software architecture description supporting component deployment and system runtime reconfiguration. In: **Proceedings of the 9th International Workshop on Component-oriented Programming (WCP'04)**. [S.l.: s.n.], 2004. 1.1, 7
- [20] BATISTA, T. V.; JOOLIA, A.; COULSON, G. Managing dynamic reconfiguration in component-based systems. In: **Proceedings of the 2nd European Workshop Software Architecture (EWSA'05)**. [S.l.]: Springer, 2005. (Lecture Notes in Computer Science, v. 3527), p. 1–17. ISBN 3-540-26275-X. 1.1, 2.3.2, 7
- [21] HALL, R. S. et al. An architecture for post-development configuration management in a wide-area network. In: **Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS'97)**. Washington, DC, USA: IEEE Computer Society, 1997. p. 269. ISBN 0-8186-7813-5. 2.1, 4.1
- [22] ARSHAD, N.; HEIMBIGNER, D.; WOLF, A. L. Deployment and dynamic reconfiguration planning for distributed software systems. **Software Quality Control**, Kluwer Academic Publishers, Hingham, MA, USA, v. 15, n. 3, p. 265–281, 2007. ISSN 0963-9314. 2.1

- [23] COUPAYE, T.; ESTUBLIER, J. Foundations of enterprise software deployment. In: **Proceedings of CSMR'00: Conference on Software Maintenance and Reengineering**. Washington, DC, USA: IEEE Computer Society, 2000. p. 65–74. ISBN 0-7695-0546-5. 2.1
- [24] LAU, K.-K.; WANG, Z. Software component models. **IEEE Transactions on Software Engineering**, IEEE Computer Society, Los Alamitos, CA, USA, v. 33, n. 10, p. 709–724, 2007. ISSN 0098-5589. 2.2.2, 2.2.3, 2, 2.4, 3.3
- [25] SUN MICROSYSTEMS. **JavaBeans 1.01 Specification**. Santa Clara, USA, ago. 1997. <http://java.sun.com/javase/technologies/desktop/javabeans/docs/spec.html>. 2.3.1
- [26] SUN MICROSYSTEMS. **JMX: Java Management Extensions**. Santa Clara, USA, out. 2006. <http://java.sun.com/javase/6/docs/technotes/guides/jmx/>. 2.3.1
- [27] OBJECT MANAGEMENT GROUP. **Common Object Request Broker Architecture: Core Specification - Version 2.6**. Needham, USA, dez. 2001. Document: formal/2001-12-01. 2.3.2, 2.3.3, 3
- [28] GARLAN, D.; MONROE, R. T.; WILE, D. Acme: Architectural description of component-based systems. In: LEAVENS, G. T.; SITARAMAN, M. (Ed.). **Foundations of Component-Based Systems**. [S.l.]: Cambridge University Press, 2000. p. 47. 2.3.2
- [29] JOOLIA, A. et al. Mapping adl specifications to an efficient and reconfigurable runtime component platform. In: **Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)**. Washington, DC, USA: IEEE Computer Society, 2005. p. 131–140. ISBN 0-7695-2548-2. 2.3.2
- [30] OBJECT MANAGEMENT GROUP. **Common Object Request Broker Architecture: Core Specification - Version 3.0**. Needham, USA, dez. 2002. Document: formal/2002-12-06. 2.3.3
- [31] WANG, N.; SCHMIDT, D. C.; O'RYAN, C. Overview of the CORBA Component Model. In: _____. **Component-based Software Engineering: Putting the Pieces Together**. Boston, USA: Addison-Wesley Longman Publishing Co., Inc., 2001. p. 557–571. ISBN 0-201-70485-4. 2.3.3

- [32] NARDI, A. R. **Componentes CORBA**. Dissertação (Mestrado) — Universidade de São Paulo, set. 2003. (document), 2.5, 2.6, 2.3.3
- [33] OBJECTWEB. **The Open CORBA Component Model Platform**. jul. 2005. Disponível em: <<http://openccm.objectweb.org>>. 2.3.3
- [34] BRICLET, F.; CONTRERAS, C.; MERLE, P. OpenCCM : une infrastructure à composants pour le déploiement d'applications à base de composants CORBA. In: IMAG/LSR (Ed.). **Conf'erece Francophone sur le D'eploiement et la (Re)Configuration de Logiciels (DECOR'04)**. [s.n.], 2004. (ISBN : 2-7261-1276-5), p. 101–112. Disponível em: <<http://hal.archives-ouvertes.fr/hal-00003287/en>>. 2.3.3, 2.3.3
- [35] SCHMIDT, D. C. **CIAO: Component-Integrated ACE ORB**. out. 2006. Disponível em: <<http://www.dre.vanderbilt.edu/CIAO>>. 2.3.3, 2.3.4
- [36] KEBBAL, D.; BERNARD, G. Component search service and deployment of distributed applications. **Distributed Objects and Applications**, IEEE Computer Society, Los Alamitos, USA, v. 00, p. 0125, 2001. 2.3.3
- [37] OBJECT MANAGEMENT GROUP. **CORBA Component Model Specification - Version 4.0**. Needham, EUA, abr. 2006. Document: formal/2006-04-01. 2.3.4
- [38] SCHMIDT, D.; KUHNS, F. An overview of the real-time CORBA specification. **IEEE Computer**, v. 33, n. 6, p. 56–63, jun. 2000. ISSN 0018-9162. 2.3.4
- [39] SCHMIDT, D. C. **TAO: The ACE ORB**. jun. 2007. Disponível em: <<http://www.dre.vanderbilt.edu/TAO>>. 4
- [40] PÉREZ, C. **Contribution à la définition et la mise en oeuvre d'un modèle de programmation à base de composants logiciels pour la programmation des grilles informatiques**. France: [s.n.], 2006. Habilitation à diriger des recherches - Université de Rennes I. 2.3.5
- [41] CUDENNEC, L.; ANTONIU, G.; BOUGÉ, L. CoRDAGE: towards transparent management of interactions between applications and resources. In: MICHAEL GERNDT AND JESUS LABARTA AND BARTON MILLER. **Proceedings of the International Workshop on Scalable**

- Tools for High-End Computing (STHEC'08)**. Kos Grèce, 2008. p. 13–24. Disponível em: <<http://hal.inria.fr/inria-00288339/en/>>. (document), 2.3.6, 2.10, 2.4
- [42] AUGUSTO, C. E. L. **Uma Infra-Estrutura para a Execução Distribuída de Componentes de Software**. Dissertação (Mestrado) — PUC-Rio, Brasil, 2008. 3, 3.1, 3.2, 3.3, 4.4, 4.4, 5.7, 6, 6.1, 7
- [43] TECGRAF. **SCS: Software Component System**. nov. 2008. Disponível em: <<http://www.tecgraf.puc-rio.br/~scorrea/scs>>. 3
- [44] BOX, D. **Essential COM**. Boston, USA: Addison-Wesley Longman Publishing Co., Inc., 1997. Foreword By-Grady Booch and Foreword By-Charlie Kindel. ISBN 0201634465. 3.1
- [45] IERUSALIMSKY, R. **Programming in Lua**. Second edition. [S.l.]: Lua.org, 2006. ISBN 8590379825. 3.1, 4.2
- [46] MAIA, R.; CERQUEIRA, R.; CALHEIROS, R. OiL: An object request broker in the Lua language. In: **Proceedings of SBRC'07 - Salão de Ferramentas**. Porto Alegre, Brazil: SBC, 2006. p. 1439–1446. 3.1, 5.1.2
- [47] HOEK, A. van der; WOLF, A. L. Software release management for component-based software. **Software – Practice & Experience**, John Wiley & Sons, Inc., New York, NY, USA, v. 33, n. 1, p. 77–98, 2003. ISSN 0038-0644. 4.1, 4.1, 4.5
- [48] HOEK, A. van der et al. Software release management. **SIGSOFT Software Engineering Notes**, ACM, New York, NY, USA, v. 22, n. 6, p. 159–175, 1997. ISSN 0163-5948. 4.1
- [49] DEBIAN. **Debian policy manual**. 2009. <http://www.debian.org/doc/debian-policy/index.html>. Último acesso em 06/2009. 4.1, 4.2
- [50] BAILEY, E. **Maximum RPM**. [S.l.]: Red Hat Inc, 2000. 442 p. <http://www.rpm.org/max-rpm>. ISBN 1-888172-78-9. 4.1, 4.2
- [51] LEONARD, T. **Zero Install Project**. jun. 2009. Disponível em: <<http://0install.net>>. 4.2
- [52] DOLSTRA, E. **Nix Project**. set. 2008. Disponível em: <<http://www.nix.org>>. 4.2
- [53] STANDARD for information technology - portable operating system interface (POSIX). Shell and utilities. **IEEE Std 1003.1, 2004 Edition**.

- The Open Group Technical Standard. Base Specifications, Issue 6. Includes IEEE Std 1003.1-2001, IEEE Std 1003.1-2001/Cor 1-2002 and IEEE Std 1003.1-2001/Cor 2-2004. Shell and Utilities**, 2004. 4.2
- [54] RED HAT. **Cygwin Project**. jan. 2009. Disponível em: <<http://www.redhat.com/services/custom/cygwin>>. 4.2
- [55] HIETANIEMI, J. **CPAN Project**. jul. 2009. Disponível em: <<http://www.cpan.org>>. 4.2
- [56] RUBYGEMS Manuals. 2009. Disponível em: <<http://www.rubygems.org>>. 4.2
- [57] LUAROCKS Project. abr. 2009. Disponível em: <<http://www.luarocks.org>>. 4.2
- [58] LUAROCKS. **Rockspeg Format**. abr. 2009. Disponível em: <http://www.luarocks.org/en/Rockspeg_format>. 4.2.1
- [59] APACHE. **The Apache ANT Project**. jan. 2010. Disponível em: <<http://ant.apache.org>>. 4.3
- [60] IONA TECHNOLOGIES. **The corbaloc and corbaname URLs**. [S.l.], ago. 2003. <http://www.iona.com/devcenter/corba/corbaloc.pdf>. 5.1
- [61] FONSECA, E. **Monitorando o Ambiente de Execução de Componentes de Software Distribuídos**. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil, 2009. 6, 6.2
- [62] DEAN, J.; GHEMAWAT, S. Mapreduce: Simplified data processing on large clusters. In: **Proceedings of OSDI 2004**. Berkeley, USA: USENIX Association, 2004. p. 137–150. (document), 6, 6.2
- [63] FONSECA, E.; CORREA, S.; CERQUEIRA, R. Experimenting middleware-level monitoring facilities to observe component-based applications. In: **Proceedings of SBCARS 2008**. Porto Alegre, Brasil: EDIPUCRS, 2008. p. 96–106. ISBN 978-85-7430-786-2. 7
- [64] TSENG, T.-L. B. et al. Applying genetic algorithm for the development of the components-based embedded system. **Computer Standards & Interfaces**, v. 27, n. 6, p. 621 – 635, 2005. ISSN 0920-5489.

Disponível em: <<http://www.sciencedirect.com/science/article/B6TYV-4F5S6C2-1/2/47d01c7228b5c5fa4f02db5227be3171>>. 7

A

Apêndice

A.1

IDL do SCS integrado ao LuaRocks

O Código A.1 apresenta os tipos de dados usados pelo *framework* do SCS e retornado pelos métodos das interfaces básicas do seu modelo de componentes. O modelo de componentes, por sua vez, possui suas interfaces ilustradas no Código A.2.

Dentre os componentes da infraestrutura de execução do SCS apenas o *ExecutionNode* e *Repository* sofreram mudanças em suas IDL, que são apresentada no Código A.4 e A.5, respectivamente. As interfaces usadas pelo componente do contêiner é apresentada no Código A.3.

A.2

IDL dos Descritores Propostos

A integração com o LUAROCKS e o desenvolvimento dos serviços de implantação motivaram a criação de dois descritores: um para indicar a lógica estrutural do componente e outro para indicar os artefatos que o implementam. O Código A.6 apresenta a definição integral desses descritores.

A.3

IDL dos Serviços para Implantação

Por fim, o Código A.7 apresenta as entidades virtuais que são manipuladas em tempo de planejamento e que existem através do uso do serviço *DeployManager*. Por outro lado, as interfaces dos serviços *DeployManager* e *Packager* são apresentadas integralmente no Código A.8.

A.4

Algoritmos do Mecanismo de Mapeamento Automático

O mecanismo que permite o mapeamento em mais alto nível considerando níveis graduais de detalhamento completa o mapeamento dos componentes na infraestrutura de execução do SCS e nas máquinas que serão usadas. O Código A.9 apresenta a implementação do método *autofit_container* que

associa um componente a um contêiner caso o usuário não tenha especificado qual contêiner usar no plano de implantação. O Código A.10 apresenta a implementação do método *autofit_exnode* que associa um contêiner a um nó de execução caso o usuário não tenha especificado qual nó de execução utilizar no plano de implantação. E, por sua vez, o Código A.11 apresenta a implementação do método *autofit_host* que associa um nó de execução a uma máquina caso o usuário não tenha especificado qual máquina utilizar no plano de implantação. Esses métodos são parametrizados pela lista de implementações disponíveis para um dado componente e por restrições de linguagem e plataforma. Portanto, caso deseje-se estender, ou mesmo alterar, a estratégia de mapeamento automático basta reimplementar esses métodos que fazem parte da implementação do serviço *DeployManager*.

Código A.1: Definição dos tipos usados pela API básica do Modelo de Componentes do SCS

```

1 #ifndef SCSDSCS_IDL
2 #define SCSDSCS_IDL
3
4 module scs {
5     module core {
6         exception StartupFailed {};
7         exception ShutdownFailed {};
8         exception InvalidName {
9             string name;
10        };
11        exception InvalidConnection {};
12        exception AlreadyConnected {};
13        exception ExceededConnectionLimit {};
14        exception NoConnection {};
15
16        typedef unsigned long ConnectionId;
17        typedef sequence<string> NameList;
18
19        struct FacetDescription {
20            string name;
21            string interface_name;
22            Object facet_ref;
23        };
24        typedef sequence<FacetDescription> FacetDescriptions;
25
26        struct ConnectionDescription {
27            ConnectionId id;
28            Object objref;
29        };
30        typedef sequence<ConnectionDescription> ConnectionDescriptions;
31
32        struct ReceptacleDescription {
33            string name;
34            string interface_name;
35            boolean is_multiplex;
36            ConnectionDescriptions connections;
37        };
38        typedef sequence<ReceptacleDescription> ReceptacleDescriptions;
39
40        struct ComponentId {
41            string name;
42            octet major_version;
43            octet minor_version;
44            octet patch_version;
45        };
46        typedef sequence<ComponentId> ComponentIdSeq;
47    };
48 };
49 #endif

```

Código A.2: Interfaces do Modelo de Componentes do SCS

```

1 #include "scsdescs.idl"
2
3 module scs { module core {
4   interface IComponent {
5     void startup() raises (StartupFailed);
6     void shutdown() raises (ShutdownFailed);
7
8     Object getFacet (in string facet_interface);
9     Object getFacetByName (in string facet);
10    ComponentId GetComponentId ();
11  };
12
13  interface IReceptacles {
14    ConnectionId connect (in string receptacle, in Object obj)
15      raises (InvalidName, InvalidConnection, AlreadyConnected,
16             ExceededConnectionLimit);
17    void disconnect (in ConnectionId id)
18      raises (InvalidConnection, NoConnection);
19    ConnectionDescriptions getConnections (in string receptacle)
20      raises (InvalidName);
21  };
22
23  interface IMetaInterface {
24    FacetDescriptions getFacets();
25    FacetDescriptions getFacetsByName(in NameList names)
26      raises (InvalidName);
27    ReceptacleDescriptions getReceptacles();
28    ReceptacleDescriptions getReceptaclesByName(in NameList names)
29      raises (InvalidName);
30  };
31 };};

```

Código A.3: Interfaces do contêiner da Infraestrutura de Execução do SCS

```

1 #include "scs.idl"
2 #include "deployment_descriptions.idl"
3
4 module scs {   module container {
5     typedef sequence<unsigned long> InterceptorIds;
6
7     struct ComponentHandle {
8         core::IComponent cmp;
9         core::ComponentId id;
10        unsigned long instance_id;
11    };
12    typedef sequence<ComponentHandle> ComponentHandleSeq;
13
14    exception ComponentNotFound{};
15    exception ComponentAlreadyLoaded{};
16    exception LoadFailure{};
17    exception InterceptorNotInstalled{};
18    exception ListLockFail{};
19    exception ContainerHalted{};
20
21    interface ComponentLoader {
22        ComponentHandle load (in core::ComponentId id, in StringSeq args)
23            raises (ComponentNotFound, ComponentAlreadyLoaded, LoadFailure);
24        void unload (in ComponentHandle handle)
25            raises (ComponentNotFound);
26    };
27
28    interface ComponentCollection {
29        ComponentHandleSeq GetComponent (in core::ComponentId id);
30        ComponentHandleSeq getComponents ();
31    };
32
33    interface ComponentInterception {
34        // 0 in position counts as end of the list.
35        // The same goes for positions above the list's current size.
36        ComponentHandle loadInterceptor (in core::ComponentId id,
37            in StringSeq args, in unsigned long position,
38            in string type)
39            raises (ListLockFail, ComponentNotFound,
40                ComponentAlreadyLoaded, LoadFailure);
41        void unloadInterceptor (in ComponentHandle handle)
42            raises (InterceptorNotInstalled, ListLockFail, ComponentNotFound);
43        void changePosition (in unsigned long instance_id,
44            in unsigned long position)
45            raises (InterceptorNotInstalled, ListLockFail, ComponentNotFound);
46        unsigned long getInterceptorPosition (in unsigned long instance_id)
47            raises (InterceptorNotInstalled, ListLockFail, ComponentNotFound);
48        InterceptorIds getClientInterceptorsOrder ();
49        InterceptorIds getServerInterceptorsOrder ();
50    };
51
52    interface ComponentSuspension {
53        void suspend ();
54        void halt ();
55        void resume ();
56        long getStatus ();
57        // status: 0 means no suspension
58        //         positive number means a halt (launches exception)
59        //         negative number means a suspension (yields coroutines)
60    };
61 };};

```

Código A.4: Interfaces do nó de execução da Infraestrutura de Execução do SCS

```

1 #include "scs.idl"
2 #include "container.idl"
3 #include "deployment_descriptions.idl"
4
5 module scs {    module execution_node {
6   typedef sequence<StringSeq> ArgsSeq;
7
8   exception ContainerAlreadyExists {};
9   exception InvalidContainer {};
10  exception RequirementNotMet{ string reason; };
11  exception InvalidProperty{ string reason; };
12  exception ComponentNotInstalled{ string reason; };
13
14  struct Property{
15    string name;
16    string value;
17    boolean read_only;
18  };
19  typedef sequence<Property> PropertySeq;
20
21  struct ContainerDescription {
22    core::IComponent container;
23    string container_name;
24    core::IComponent execution_node;
25  };
26  typedef sequence<ContainerDescription> ContainerDescriptionSeq;
27
28  interface ExecutionNode {
29    core::IComponent startContainer (in string container_name ,
30                                     in PropertySeq props)
31      raises (ContainerAlreadyExists , InvalidProperty);
32    void killContainer(in string container_name)
33      raises (core::InvalidName);
34    core::IComponent getContainer (in string container_name);
35    ContainerDescriptionSeq getContainers ();
36    string getName();
37    string getPlatform();
38  };
39
40  interface ContainerManager {
41    void registerContainer(in string name, in core::IComponent ctr ,
42                          in string pid)
43      raises (ContainerAlreadyExists , InvalidContainer);
44    void unregisterContainer(in string name) raises (core::InvalidName);
45  };
46
47  interface Installer {
48    void install(in core::ComponentId id , in string lang)
49      raises (ComponentNotInstalled);
50    void uninstall(in core::ComponentId id , in string lang);
51    StringSeq getEnvVars(in string lang);
52    string getRootPath(in string lang);
53    StringSeq getDependenciesPath(in core::ComponentId id ,
54                                 in string lang);
55    string getInstallPath(in core::ComponentId id , in string lang);
56    core::ComponentIdSeq getInstalledComponents(in string lang);
57    deployer::ComponentDescription getDescription(in core::ComponentId id ,
58                                                  in string lang)
59      raises (ComponentNotInstalled);
60  };
61 };};

```

Código A.5: Interfaces do repositório da Infraestrutura de Execução do SCS

```

1 #include "deployment_descriptions.idl"
2
3 module scs { module repository {
4   exception ComponentAlreadyUploaded {};
5   exception ComponentNotUploaded {};
6
7   struct Constraint {
8     string key;
9     string value;
10  };
11  typedef sequence<Constraint> ConstraintSeq;
12
13  interface ComponentRepository {
14    void upload (in deployer::Package pkg, in string help_info)
15      raises (ComponentAlreadyUploaded, ComponentNotUploaded);
16
17    void delete (in core::ComponentId id, in string lang, in string plat)
18      raises (ComponentNotUploaded);
19
20    void copy (in deployer::ComponentDescription desc,
21              in ComponentRepository repo)
22      raises (ComponentAlreadyUploaded, ComponentNotUploaded);
23
24    deployer::Package download (in core::ComponentId id,
25                               in string lang, in string plat)
26      raises (ComponentNotUploaded);
27
28    deployer::ComponentDescriptionSeq search (
29      in ConstraintSeq constraints);
30    deployer::ComponentDescriptionSeq searchById (
31      in core::ComponentId id);
32
33    deployer::ComponentDescription getDescription (
34      in core::ComponentId id,
35      in string lang, in string plat)
36      raises (ComponentNotUploaded);
37
38    deployer::ComponentDescriptionSeq getAllDescriptions();
39  };
40 };};

```

Código A.6: Definição dos descritores do Modelo e Implementação

```

1 #include "scs.idl"
2
3 module scs {
4     typedef sequence<string> StringSeq;
5     typedef sequence<octet> OctetSeq;
6
7     module deployer {
8         struct PhysicalResources {
9             unsigned long cpu;
10            unsigned long mem;
11            string os;
12            string arch;
13            unsigned long bandwidth;
14        };
15
16        struct HostDescription {
17            string name;
18            string ip;
19            unsigned long port;
20            PhysicalResources resources;
21            StringSeq languages;
22        };
23        typedef sequence<HostDescription> HostSeq;
24
25        struct Facet {
26            string name;
27            string interface_name;
28        };
29        typedef sequence<Facet> FacetSeq;
30
31        struct Receptacle {
32            string name;
33            string interface_name;
34            boolean is_multiplex;
35        };
36        typedef sequence<Receptacle> ReceptacleSeq;
37
38        struct ComponentTemplate {
39            core::ComponentId id;
40            FacetSeq facets;
41            ReceptacleSeq receptacles;
42        };
43        typedef sequence<ComponentTemplate> ComponentTemplateSeq;
44
45        struct Artefact {
46            string name;
47            string class_name;
48        };
49        typedef sequence<Artefact> ArtefactSeq;
50
51        struct ComponentDescription {
52            core::ComponentId template_id;
53            string lang;
54            string plat;
55            string entry_point;
56            ArtefactSeq facets;
57            ArtefactSeq receptacles;
58        };
59        typedef sequence<ComponentDescription> ComponentDescriptionSeq;
60
61        struct Package {
62            ComponentDescription info;
63            OctetSeq file;
64        };
65        typedef sequence<Package> PackageSeq;
66    };
67 };

```


Código A.7: Definições das entidades virtuais do planejamento

```

1 #include "scs.idl"
2 #include "deployment.idl"
3 module scs {
4     module deployer {
5         exception AlreadyDeployed {}; exception NotDeployed {};
6         typedef sequence<ComponentEntity> ComponentEntitySeq;
7         typedef sequence<ContainerEntity> ContainerEntitySeq;
8         typedef sequence<ExecutionNodeEntity> ExecutionNodeEntitySeq;
9         typedef sequence<RepositoryEntity> RepositoryEntitySeq;
10        interface Deployment { //processo de implantação
11            boolean deploy() raises (AlreadyDeployed);
12            boolean deploy_connections() raises (AlreadyDeployed);
13            boolean activate() raises (NotDeployed);
14            boolean deactivate() raises (NotDeployed);
15            boolean undeploy();
16        };
17        // tipos de dados da conexão planejada
18        struct FacetEndpoint { BasicComponentEntity entity; string facetname; };
19        struct ReceptacleConnections {
20            string receptname; FacetEndpointSeq endpoints;
21        };
22        typedef sequence<FacetEndpoint> FacetEndpointSeq;
23        typedef sequence<ReceptacleConnections> ReceptacleConnectionsSeq;
24        // Entidade representante de um componente da infraestrutura
25        interface BasicComponentEntity : Deployment {
26            void set_args(in StringSeq args);
27            void set_id(in ComponentId id);
28            void set_description(in ComponentTemplate desc);
29            ComponentTemplate get_description();
30            void add_connection(in string recept,
31                               in BasicComponentEntity provider, in string facet);
32            void del_connection(in string recept,
33                               in BasicComponentEntity provider, in string facet);
34            ReceptacleConnectionsSeq get_connections();
35            void set_property(in string name, in string value);
36            string get_property(in string name);
37            string get_nickname();
38            Plan get_plan();
39            Object get_facet(in string facetname) raises (NotDeployed);
40        };
41        // Entidade representante dos componentes do usuário
42        interface ComponentEntity : BasicComponentEntity {
43            void set_container(in ContainerEntity container);
44            ContainerEntity get_container();
45        };
46        // Entidade representante do contêiner
47        interface ContainerEntity : BasicComponentEntity {
48            void set_node(in ExecutionNodeEntity node);
49            ExecutionNodeEntity get_node();
50        };
51        // Entidade representante do nó de execução
52        interface ExecutionNodeEntity : BasicComponentEntity {
53            void set_host(in HostDescription host);
54            HostDescription get_host();
55            void add_known_repositories(in RepositoryEntity repo);
56            void del_known_repositories(in RepositoryEntity repo);
57        };
58        // Entidade representate do repositório
59        interface RepositoryEntity : BasicComponentEntity {
60            void set_host(in HostDescription host);
61            HostDescription get_host();
62            void set_privacy(in boolean private) raises (AlreadyDeployed);
63            boolean get_privacy();
64            void publish_desc(in ComponentTemplate structure);
65            void publish_pkg(in Package pkg);
66            void unpublish_desc(in core::ComponentId id);
67            void unpublish_pkg(in core::ComponentId id,
68                               in string lang, in string plat);
69            ComponentTemplate get_desc(in core::ComponentId id);
70            Package get_pkg(in core::ComponentId id,
71                            in string lang, in string plat);
72            ComponentDescriptionSeq search_byid(in core::ComponentId id);
73    };};};

```

Código A.8: Interfaces dos Serviços DeployManager e Packager

```

1 #include "scs.idl"
2 #include "deployer_entities.idl"
3
4 module scs { module deployer {
5     exception InternalProblem { string reason; };
6     exception InvalidPackage { string reason; };
7     exception BuildError { string reason; };
8     exception PublishError {};
9
10    // Plano de implantação
11    interface Plan : Deployment {
12        // planejamento
13        ComponentEntity create_component();
14        ContainerEntity create_container(in string lang);
15        ExecutionNodeEntity create_exnode();
16        RepositoryEntity create_repository();
17        // inspeção
18        ComponentEntitySeq get_components();
19        ContainerEntitySeq get_containers();
20        ExecutionNodeEntitySeq get_nodes();
21        RepositoryEntitySeq get_repositories();
22        Manager get_manager();
23        void mark_as_admin(in boolean flag);
24        ContainerEntitySeq less_used_containers();
25        ExecutionNodeEntitySeq less_used_nodes();
26        HostSeq less_used_hosts();
27        void set_max_container_usage(in long max);
28        long get_max_container_usage();
29        void set_max_node_usage(in long max);
30        long get_max_node_usage();
31    };
32    typedef sequence<Plan> PlanSeq;
33
34    // Serviço para implantação distribuída
35    interface Manager {
36        Plan create_plan();
37        void delete_plan(in Plan p);
38        PlanSeq get_plans();
39        // configuração entre componentes
40        boolean connect_component(in ComponentEntity consumer,
41            in string recept, in ComponentEntity provider,
42            in string facet);
43        // compartilhando repositórios públicos entre planos
44        void add_public_repositories(in RepositoryEntity repo)
45            raises (InternalProblem);
46        void del_public_repositories(in RepositoryEntity repo)
47            raises (InternalProblem);
48        RepositoryEntitySeq get_public_repositories()
49            raises (InternalProblem);
50        // registro de recursos
51        HostSeq get_hosts();
52    };
53    // Serviço para empacotamento
54    interface Packager {
55        Package create(in OctetSeq rockspec_stream, in OctetSeq impl_stream,
56            out ComponentTemplate tmpl)
57            raises (InvalidPackage, BuildError);
58        Package create_from_dir(in string rockspec_file, in string dirname,
59            out ComponentTemplate tmpl)
60            raises (InvalidPackage, BuildError);
61    };
62 };};

```

Código A.9: Algoritmo de decisão para associar um componente a um contêiner automaticamente

```

1 function ComponentEntity:autofit_container(available_implementations)
2   local found,node,language
3   local plan = self.plan
4   local existent_containers = self:less_used_containers()
5   for i, container in ipairs(existent_containers) do
6     if plan:get_score(container) < plan:get_max_container_usage() then
7       language = container:get_property("language")
8       if available_implementations[language] then
9         node = container:get_node()
10        if not node then —nó de execução não tinha sido especificado
11          node = self:autofit_exnode(available_implementations, language)
12        end
13        platform = node:get_property("platform")
14        if available_implementations[language][platform] then
15          found = container
16          if not container:get_node() then
17            container:set_node(node)
18          end
19          break
20        end
21      end
22    else
23      —continuamos buscando algum container existente que possa ser usado
24    end
25  end
26  —não achamos nenhum container compatível, então vamos criar um novo
27  if not found then
28    lang_constraint = next(available_implementations)
29    found = plan:create_container(lang_constraint)
30    node = self:autofit_exnode(available_implementations, lang_constraint)
31    found:set_exnode(node)
32  end
33  return found
34 end

```

Código A.10: Algoritmo de decisão para associar um contêiner a um nó de execução automaticamente

```

1  function ComponentEntity:autofit_exnode(available_implementations ,
2                                     lang_constraint)
3
4  local plan = self.plan
5  local found,host,platform
6  local existent_nodes = self:less_used_nodes()
7  for i, node in ipairs(existent_nodes) do
8      if plan:get_score(node) < plan:get_max_node_usage() then
9          platform = node:get_property("platform")
10         if not platform then
11             host = self:autofit_host(available_implementations ,
12                                     lang_constraint, plat_constraint)
13             platform = plat_constraint or (host.resources.os..
14                                             "-" .. host.resources.arch)
15         else
16             host = node:get_host()
17         end
18         for j, supported_lang in ipairs(host.languages) do
19             if (supported_lang == lang_constraint) and
20                 (available_implementations[supported_lang][platform])
21                 then
22                 found = node
23                 if not node:get_host() then
24                     node:set_host( host )
25                 end
26                 break
27             end
28         end
29         —continuamos buscando algum nó de execução que possa ser usado
30     end
31 end
32 —não achamos nenhum nó de execução compatível, então vamos usar um outro
33 if not found then
34     platform_list = next(available_implementations[lang_constraint])
35     plat_constraint = next(platform_list)
36     found = plan:create_exnode()
37     host = self:autofit_host(available_implementations ,
38                             lang_constraint, plat_constraint)
39     found:set_host(host)
40 end
41 return found
42 end

```

Código A.11: Algoritmo de decisão para associar um nó de execução a uma máquina automaticamente

```
1 function ComponentEntity:autofit_host(available_implementations ,  
2                                     lang_constraint , plat_constraint)  
3   local found , platform  
4   local existent_hosts = self:less_used_hosts()  
5   for i , host in ipairs(existent_hosts) do  
6     platform = host.resources.os .. "-" .. host.resources.arch  
7     —só existe 1 'nó de execução' por 'host'  
8     if platform == plat_constraint then  
9       for i , language in ipairs(host.languages) do  
10        if language == lang_constraint then  
11          found = host  
12          break  
13        end  
14      end  
15      if found then break end  
16    end  
17  end  
18  if not found then  
19    error("Não existe máquina para a plataforma solicitada:",  
20          plat_constraint)  
21  end  
22  return found  
23 end
```