6
Referências Bibliográficas

[1] BROOKS, F. **The Mythical Man-Month**. [S.l.]: Addison-Wesley Pub. Co, 1995. ISBN 0201835959. 1, 5

[2] FIELDING, R. T.; TAYLOR, R. N. Principled design of the modern web architecture. **ACM Trans. Internet Technol.**, ACM, New York, NY, USA, v. 2, n. 2, p. 115–150, 2002. ISSN 1533-5399. 1

[3] FIELDING, R. et al. **Hypertext Transfer Protocol – HTTP/1.1**. United States: RFC Editor, 1999. 1

[4] MEIJER, E.; SZYPERSKI, C. Overcoming independent extensibility challenges. **Commun. ACM**, ACM, New York, NY, USA, v. 45, n. 10, p. 41–44, 2002. ISSN 0001-0782. 1

[5] SZYPERSKI, C. Independently extensible systems - software engineering potential and challenges -. In: **In Proceedings of the 19th Australasian Computer Science Conference**. [S.l.: s.n.], 1996. 1

[6] SZYPERSKI, C. **Component Software: Beyond Object-Oriented Programming**. Boston, MA, USA: Addison-Wesley Longman, 2002. ISBN 0201745720. 1

[7] BRUNETON, E. et al. The fractal component model and its support in java. **Software: Practice and Experience**, Wiley Online Library, v. 36, n. 11-12, p. 1257–1284, 2006. ISSN 1097-024X. 1, 2.1

[8] OBJECT MANAGEMENT GROUP. **CORBA Components - Version 3.0**. Needham, USA, jun. 2002. Document: formal/2002-06-65. 1, 2.1

[9] MICROSOFT. **COM - Componet Object Model Technology**. Disponível em: <http://www.microsoft.com/com>. 1, 2.1

[10] COULSON, G. et al. A component model for building systems software. In: HAMZA, M. H. (Ed.). **IASTED Conf. on Software Engineering**

and Applications. [S.l.]: IASTED/ACTA Press, 2004. p. 684–689. ISBN 0-88986-425-X. 1, 2.1

[11] MAIA, R.; CERQUEIRA, R.; RODRIGUEZ, N. de L. R. An infrastructure for development of dynamically adaptable distributed components. In: MEERSMAN, R.; TARI, Z. (Ed.). **CoopIS/DOA/ODBASE (2)**. [S.l.]: Springer, 2004. (Lecture Notes in Computer Science, v. 3291), p. 1285–1302. ISBN 3-540-23662-7. 1, 2.1

[12] MAIA, R. **Um Framework para Adaptação Dinâmica de Sistemas Baseados em Componentes Distribuídos**. Dissertação (Mestrado) — Catholic University of Rio de Janeiro (PUC-Rio), March 2004. (in portuguese). Disponível em: <http://www.inf.puc-rio.br/ maia/papers/maia04framework.pdf>. 1, 2.1

[13] AUGUSTO, C. et al. **SCS: Software Component System**. maio 2006. Disponível em: <http://www.tecgraf.puc-rio.br/scs>. 1, 2.1, 2.1.1, 3.4, 4, 5

[14] MEYER, B. **Object-Oriented Software Construction, Second Edition**. [S.l.]: Prentice Hall, 2000. ISBN 0136291554. 1, 3.5, 5

[15] AJMANI, S.; LISKOV, B.; SHRIRA, L. Modular software upgrades for distributed systems. In: **European Conference on Object-Oriented Programming (ECOOP)**. [S.l.: s.n.], 2006. 1, 2.3, 3.4, 3.4, 5, C

[16] AJMANI, S. **Automatic Software Upgrades for Distributed Systems**. Tese (Ph.D.) — MIT, set. 2004. Also as Technical Report MIT-LCS-TR-1012. 1, 2.3, 3.4, 3.4, 5, C

[17] TAYLOR, R. N.; MEDVIDOVIC, N.; OREIZY, P. Architectural styles for runtime software adaptation. **Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture**, 2009. 1

[18] AJMANI, S. A review of software upgrade techniques for distributed systems. August 2002. Disponível em: <http://www.pmg.lcs.mit.edu/ ajmani/papers/review.pdf>. 1

[19] AUGUSTO, C. E. L. **Uma Infra-Estrutura para a Execução Distribuída de Componentes de Software**. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ, Brasil, set. 2008. 2.1.1

[20] IERUSALIMSCHY, R. **The Programming Language Lua**. fev. 2008. Disponível em: <http://www.lua.org/>. 2.1.1, 3.4

[21] GOSLING, J. et al. **Java (TM) Language Specification, The (Java (Addison-Wesley))**. [S.l.]: Addison-Wesley Professional, 2005. ISBN 0321246780. 2.1.1

[22] STROUSTRUP, B. **The C++ Programming Language, Third Edition**. [S.l.]: Addison-Wesley, 2000. ISBN 0201700735. 2.1.1

[23] PETZOLD, C. **Programming Windows with C# (Core Reference)**. [S.l.]: Microsoft Press Redmond, WA, USA, 2001. ISBN 0735613702. 2.1.1

[24] OBJECT MANAGEMENT GROUP. **The Common Object Request Broker Architecture (CORBA) Specification - Version 3.1**. [S.l.], jan. 2008. Document: formal/2008-01-04. 2.1.1

[25] HENNING, M. A New Approach to Object-Oriented Middleware. **IEEE Internet Computing**, v. 8, n. 1, p. 66–75, 2004. 2.2

[26] HENNING, M.; SPRUIELL, M. **Distributed Programming with Ice**. 2008. Http://www.zeroc.com/download/Ice/3.3/Ice-3.3.0.pdf (Last Visited in 08/06/2008). 2.2, 2.2

[27] HENNING, M. The rise and fall of corba. **ACM Queue**, ACM, New York, USA, v. 4, n. 5, p. 28–34, jun. 2006. ISSN 1542-7730. 2.2

[28] HICKS, M.; NETTLES, S. M. Dynamic software updating. **ACM Transactions on Programming Languages and Systems (TOPLAS)**, v. 27, n. 6, p. 1049–1096, November 2005. 2.3.1

[29] SUBRAMANIAN, S.; HICKS, M.; MCKINLEY, K. S. Dynamic software updates: A VM-centric approach. In: **Proceedings of the ACM Conference on Programming Language Design and Implementation (PLDI)**. [S.l.: s.n.], 2009. p. 1–12. 2.3.1

[30] OPENBUS. **Openbus - Enterprise Integration Application Middleware**. 2006. Disponível em: <http://www.tecgraf.puc-rio.br/openbus>. 3.1, 4

[31] GAMMA, E. et al. **Design patterns: elements of reusable object-oriented software**. [S.l.]: Addison-wesley Reading, MA, 1995. 3.4

[32] JUNIOR, A. A. B. **Implantação de Componentes de Software Distribuídos Multi-Linguagem e Multi-Plataforma**. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ, Brasil, ago. 2009. 5

[33] SALDANHA, H. M. de C. **Utilizando Anotações em Linguagens Orientadas a Objetos para Suporte à Programação Orientada a Componentes**. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ, Brasil, ago. 2010. 5

A
IDL das Interfaces do SCS-MV

A.1
Interface IFacet

Código A.1: IDL da interface *IFacet*.

```
1  module scs {
2    module core {
3      exception InvalidName {
4        string name;
5      };
6      exception InvalidVersion {
7        string version;
8      };
9
10     struct AccessInterface {
11       string interface_name;
12       string version;
13       Object objref;
14     };
15
16     interface IFacet {
17       string getName();
18       IFacet getComponent();
19       AccessInterfaceDescriptions getAccessInterfaces();
20       AccessInterface getAccessInterface(in string interface_name) raises (InvalidName);
21       AccessInterface getAccessInterfaceByVersion(in string version) raises (InvalidVersion);
22     };
23     typedef sequence<IFacet> Facets;
24   };
25 };
```

A.2
Interface IComponent

Código A.2: IDL da interface *IComponent*.

```
module scs {
  module core {
    exception StartupFailed {};
    exception ShutdownFailed {};
    exception InvalidName {
      string name;
    };

    struct ComponentId {
      string name;
      octet major_version;
      octet minor_version;
      octet patch_version;
      string platform_spec;
    };
    typedef sequence<ComponentId> ComponentIdSeq;

    interface IComponent {
      void startup() raises (StartupFailed);
      void shutdown() raises (ShutdownFailed);
      Facets getFacet (in string facet_interface);
      IFacet getFacetByName (in string facet) raises (InvalidName);
      ComponentId getComponentId ();
    };
    typedef sequence<IComponent> IComponentSeq;
  };
};
```

A.3
Interface IReceptacles

Código A.3: IDL da interface *IReceptacles*.

```
module scs {
  module core {
    exception InvalidName {
      string name;
    };
    exception InvalidConnection {};
    exception AlreadyConnected {};
    exception ExceededConnectionLimit {};
    exception NoConnection {};
    typedef unsigned long ConnectionId;

    struct ConnectionDescription {
      ConnectionId id;
      IFacet objref;
    };
    typedef sequence<ConnectionDescription> ConnectionDescriptions;

    struct ReceptacleDescription {
      string name;
      boolean is_multiplex;
      ConnectionDescriptions connections;
      sequence<string> versionList;
    };
    typedef sequence<ReceptacleDescription> ReceptacleDescriptions;

    interface IReceptacles {
      ConnectionId connect (in string receptacle, in IFacet obj)
         raises (InvalidName, InvalidConnection, AlreadyConnected,
                 ExceededConnectionLimit);
      void disconnect (in ConnectionId id)
         raises (InvalidConnection, NoConnection);
      ConnectionDescriptions getConnections (in string receptacle)
         raises (InvalidName);
      AccessInterfaceDescriptions getConnectionsVersions (in string receptacle)
         raises (InvalidName);
    };
  };
};
```

A.4
Interface IMetaInterface

Código A.4: IDL da interface *IMetaInterface*.

```
module scs {
  module core {
    exception InvalidName {
      string name;
    };
    typedef sequence<string> NameList;

    struct AccessInterfaceDescription {
      string interface_name;
      string version;
    };
    typedef sequence<AccessInterfaceDescription> AccessInterfaceDescriptions;

    struct FacetDescription {
      string name;
      string interface_name;
      string version;
      AccessInterfaceDescriptions versions;
    };
    typedef sequence<FacetDescription> FacetDescriptions;

    struct ReceptacleDescription {
      string name;
      boolean is_multiplex;
      ConnectionDescriptions connections;
      sequence<string> versionList;
    };
    typedef sequence<ReceptacleDescription> ReceptacleDescriptions;

    interface IMetaInterface {
      FacetDescriptions getFacets();
      FacetDescriptions getFacetsByName(in NameList names)
         raises (InvalidName);
      ReceptacleDescriptions getReceptacles();
      ReceptacleDescriptions getReceptaclesByName(in NameList names)
         raises (InvalidName);
    };
  };
};
```

B
O Openbus

O Openbus é uma arquitetura aberta e orientada a serviços (*Service-Oriented Architecture*, ou SOA) para integrar dados e aplicações heterogêneas de domínios distintos ou não. A arquitetura é baseada em um *barramento* em que os usuários se conectam para consumir ou prover serviços.

Ele é baseado no *middleware* CORBA. Seu barramento é formado por componentes que podem oferecer serviços ou podem apenas consultar serviços de outros componentes. Estes componentes são implementados através de um sistema de componentes para CORBA chamado SCS(Sistema de Componentes de Software). No SCS, os componentes são compostos por facetas que representam as interfaces que estes componentes oferecem.

A infra-estrutura do Openbus provê dois serviços básicos necessários para que o usuário possa utilizar o barramento, o serviço de acesso e o serviço de registro. Um terceiro serviço auxilia o compartilhamento de informações entre um grupo de clientes, que é o serviço de sessão. Por último, a arquitetura do Openbus disponibiliza uma interface comum para a implementação de um serviço de dados.

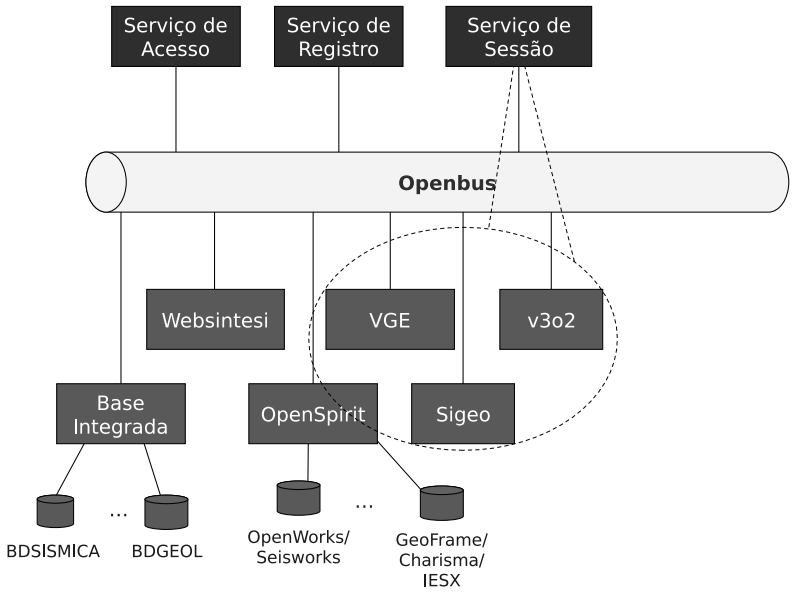

Figura B.1: Arquitetura do Openbus

Serviço de Acesso É o ponto de entrada no barramento. Sua referência é conhecida por todos. Para que um usuário possa entrar no barramento para consumir ou prover serviços, é necessário que ele se autentique no mesmo adquirindo uma credencial de acesso. A autenticação pode ser realizada através de um par usuário/senha ou através de um certificado digital. Tipicamente, aplicações "clientes"se autenticam com um par usuário/senha, que normalmente será a identificação e a senha do usuário que iniciou a aplicação. Servidores tipicamente se autenticam através de um certificado.

Após a autenticação, uma credencial é emitida para o usuário para que o mesmo possa acessar os serviços disponibilizados pelo barramento. Essa credencial é composta por uma identificador unívoco e por um nome de uma entidade à qual está associada, que pode ser um usuário ou um serviço. Essa credencial tem um ciclo de vida e deve ser renovada para que não expire.

Serviço de Registro Responsável por controlar as ofertas de serviços disponíveis no barramento. Um componente que queira oferecer um serviço deve explicitamente registrar sua oferta no serviço de registro. Componentes que desejam utilizar um serviço podem obter a localização e as propriedades de provedores desse serviço através de consultas ao serviço de registro.

Serviço de Sessão Oferece um mecanismo simplificado de troca de mensagens entre os componentes que compartilham uma mesma sessão.

Serviço de Dados Disponibilização de uma interface para a implementação de um serviço de acesso a dados hierárquicos ou não. A interface provê uma faceta para a navegação e outra para o acesso ao dado.

C
Infraestrutura de Atualização Dinâmica do Upstart

A infra-estrutura é invisível para o sistema sendo atualizado. Sua função principal é disseminar as informações sobre as atualizações dos nós do sistema, garantir que os nós atualizem no instante apropriado e permitir a interação entre eles, mesmo quando executando versões distintas.

Os componentes que constituem a infra-estrutura são: servidor de atualização (*upgrade server*), camada de atualização (*upgrade layer*), rede de distribuição de software (*software distribution network*), e banco de atualizações (*update database*). Vide a figura C.1.

O servidor de atualização armazena as configurações que identificam a menor versão ativa, o esquema inicial, e os componentes de todas as atualizações das classes. A camada de atualização do nó aplica etiquetas com o número da versão do objeto nas chamadas realizadas pelo mesmo. As chamadas realizadas por um *SO* são etiquetadas com a versão do *SO*.

A camada de atualização descobre novas versões baixando periodicamente a configuração do servidor de atualização, para verificar se existe uma atualização para a próxima versão do nó. Ela também pode descobrir novas versões através do mecanismo de fofoca [15, 16], onde analisa os números de versão das chamadas recebidas e compartilha as versões mais novas que encontrou com outros nós. Quando um novo número de versão é encontrado, então ela baixa a configuração mais nova do servidor.

É através da rede de distribuição de software que a camada de atualização obtém os componentes da atualização e a nova implementação da classe. Com isso, a camada de atualização instala o objeto futuro da atualização, e depois invoca a função de agendamento, que executa em paralelo com a versão atual do objeto e determina o instante no qual o nó deve atualizar.

Em resposta ao sinal de atualização enviado pela função de agendamento, a camada de atualização desliga o objeto corrente e todos os *SOs*, instala a nova implementação da classe e executa a função de transformação. Depois, ela descarta o objeto futuro e instala o objeto passado da versão antiga, e inicializa o objeto corrente, que recupera o estado persistente recentemente transformado. Por fim, a camada de atualização notifica o banco de atualizações que seu nó

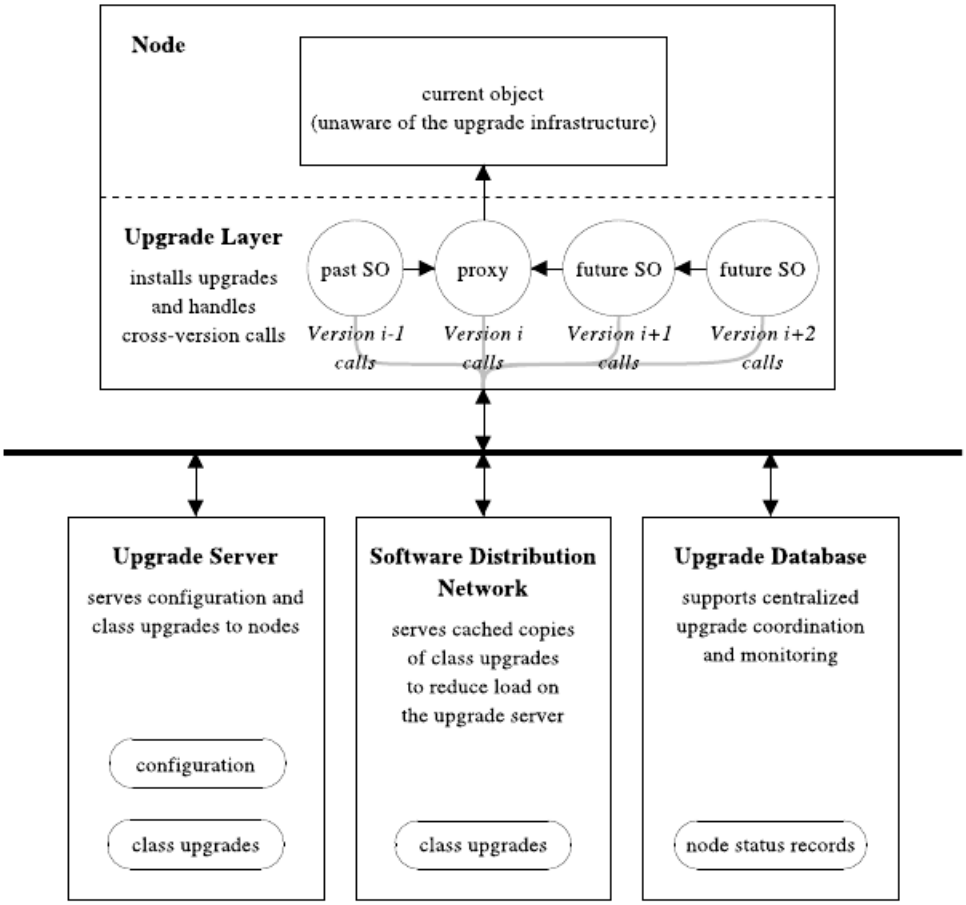

Figura C.1: A infra-estrutura de atualização. Os componentes se comunicam através de uma rede (linha em negrito); setas indicam direção de chamadas de métodos (possivelmente remotas). O nó esta executando uma instância de classe na versão *i* e SOs para as versões *i-1*, *i+1* e *i+ 2*.

está executando a nova versão.

A função de agendamento pode acessar um banco de atualizações para coordenar o agendamento da atualização com outros nós e para permitir que operadores humanos monitorem e controlem o progresso da atualização. Uma atualização define um novo esquema do sistema e um mapeamento da versão antiga de uma classe para uma nova. Todo esquema possui um número de versão associado.

Para realizar uma atualização de um nó é necessário passar um conjunto de informações composto por seis componentes:

oldClassID

Identifica a classe que agora é obsoleta.

newClassID

Identifica a classe que irá substituí-la.

TF

Identifica a função de transformação, que gera o estado inicial de persistência do objeto a partir do estado antigo.

SF

Identifica a função de agendamento, que informa ao nó quando ele deve atualizar.

pastSO

Objeto que permite ao nó suportar o antigo comportamento da classe após a atualização.

futureSO

Objeto que permite ao nó suportar o novo comportamento da classe antes da atualização.

Não é necessário especificar todos os seis componentes em todas as atualizações. A função de transformação só é necessária se ocorrer uma alteração na maneira como o objeto organiza o seu estado persistido. Os objetos de simulação só são necessários quando ocorre uma alteração do tipo do objeto, então, dependendo da alteração realizada, uma atualização pode omitir o pastSO, ou o futureSO, ou ambos. A função de agendamento não pode ser omitida, porém, ela é fácil de implementar e às vezes é possível selecionar uma de uma biblioteca.