

## 7

### Referências Bibliográficas

- [1] MCINROY, D. Mass produced software components. In: NAUR, P.; RANDELL, B. (Ed.). Brussels, Belgium: Scientific Affairs Division, NATO, 1968. p. 138–155. 1
- [2] SZYPERSKI, C. **Component Software: Beyond Object-Oriented Programming**. Boston, MA, USA: Addison-Wesley Longman, 2002. ISBN 0201745720. 1
- [3] SZYPERSKI, C. Component technology: what, where, and how? In: IEEE COMPUTER SOCIETY WASHINGTON, USA. **Proceedings of the 25th International Conference on Software Engineering**. [S.l.], 2003. p. 684–693. 1
- [4] BODOFF, S. et al. **The J2EE Tutorial, Second Edition**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2004. ISBN 032124575X. Disponível em: <<http://portal.acm.org/citation.cfm?id=1024190>>. 1
- [5] BRUNETON, E. et al. The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems. **Software Practice & Experience**, John Wiley & Sons, Inc., New York, USA, v. 36, n. 11–12, p. 1257–1284, set. 2006. ISSN 0038-0644. 1, 5.3
- [6] COULSON, G. et al. A generic component model for building systems software. **ACM Trans. Comput. Syst.**, ACM, New York, NY, USA, v. 26, n. 1, p. 1–42, 2008. ISSN 0734-2071. 1
- [7] AUGUSTO, C. et al. **SCS: Software Component System**. maio 2007. Disponível em: <<http://www.tecgraf.puc-rio.br/scorrea/scs>>. 1, 2, 2.2, 6
- [8] KICZALES, G. et al. Aspect-oriented programming. In: AKsIT, M.; MATSUOKA, S. (Ed.). **Proceedings European Conference on Object-Oriented Programming**. Berlin, Heidelberg, and New

- York: Springer-Verlag, 1997. v. 1241, p. 220–242. Disponível em: <[citeseer.ist.psu.edu/kiczales97aspectoriented.html](http://citeseer.ist.psu.edu/kiczales97aspectoriented.html)>. 1, 4.2.2, 5, 5.3
- [9] EICHBERG, M. Component-based software development with aspect-oriented programming. **Journal of Object Technology**, v. 4, n. 3, p. 21–26, abr. 2005. ISSN 1660-1769. Disponível em: <<http://www.jot.fm/contents/issue200504/article3.html>>. 1, 5
- [10] EICHBERG, M.; SCHÄFER, T.; MEZINI, M. Using annotations to check structural properties of classes. In: **FASE**. [S.l.: s.n.], 2005. p. 237–252. 1, 3, 5.1
- [11] WALLS, C.; RICHARDS, N.; OBERG, R. **XDoclet in Action (In Action series)**. Greenwich, CT, USA: Manning Publications Co., 2003. ISBN 1932394052. 1
- [12] FRANZ, M.; FRÖHLICH, P. H.; GAL, A. **Supporting Software Composition at the Programming-Language Level**. 2003. 1
- [13] CHAPPELL, D. **Introducing SCA**. jul 2007. Disponível em: <[http://www.davidchappell.com/articles/Introducing\\_SCA.pdf](http://www.davidchappell.com/articles/Introducing_SCA.pdf)>. 1, 5, 5.1, 5.3
- [14] ROUYVOY, R.; MERLE, P. Leveraging component-based software engineering with fraclet. **Annales des TÃ©lÃ©communications**, v. 64, n. 1-2, p. 65–79, 2009. Disponível em: <<http://dblp.uni-trier.de/db/journals/adt/adt64.htmlRouvoyM09>>. 1, 5, 5.2, 5.3
- [15] GOSLING, J. et al. **The Java Language Specification, Third Edition**. 3. ed. Amsterdam: Addison-Wesley Longman, 2005. 688 p. ISBN 0321246780. 1, 2
- [16] INTERNATIONAL, E. **Standard ECMA-334 - C Language Specification**. 4. ed. [s.n.], 2006. Disponível em: <<http://www.ecma-international.org/publications/standards/Ecma-334.htm>>. 1
- [17] MWLab. <http://mwlab.tecgraf.puc-rio.br/MWLab/>. 1
- [18] Tecgraf. <http://www.tecgraf.puc-rio.br/>. 1
- [19] PONTIFÍCIA Universidade Católica do Rio de Janeiro. <http://www.puc-rio.br/>. 1
- [20] APT. <http://java.sun.com/j2se/1.5.0/docs/guide/apt/>. 1, 3.3
- [21] GREEN, T. R. G. Cognitive dimensions of notations. In: **People and Computers V**. [S.l.]: University Press, 1989. p. 443–460. 1, 4, 4.2, 6

- [22] IERUSALIMSCHY, R.; CELES, W.; FIGUEIREDO, L. H. de. **Lua - The Programming Language**. <http://www.lua.org/>. 2
- [23] IERUSALIMSCHY, R. **Programming in Lua**. [S.l.]: Lua.org, 2003. 2
- [24] SZYPERSKI, C. Component technology - what, where, and how? **Software Engineering, International Conference on**, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 684, 2003. ISSN 0270-5257. 2
- [25] OMG: Object Management Group. **CORBA Component Model**. April 2006. <http://www.omg.org/technology/documents/formal/components.htm>. 2.1, 4.2.1
- [26] 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, 4.2.1
- [27] OMG - The object management group. <http://www.omg.org/>. 2.1
- [28] ECLIPSE Platform. <http://www.eclipse.org/>. 3.3
- [29] CLARKE, S.; BECKER, C. Using the Cognitive Dimensions Framework to evaluate the usability of a class library. In: PETRE, M.; BUDGEN, D. (Ed.). **15th Annual Workshop of the Psychology of Programming Interest Group**. [S.l.: s.n.], 2003. p. 359–366. 4, 6
- [30] Green, T. R. G and Blackwell. **Cognitive Dimensions of Notations Artefacts: a Tutorial**. 1998. <http://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/CDtutorial.pdf>. 4.2.1
- [31] SEINTURIER, L. et al. A component model engineered with components and aspects. In: **Proceedings of the 9th International SIGSOFT Symposium on Component-Based Software Engineering (CBSE'06)**. Väasteras, Sweden: Springer, 2006. (Lecture Notes in Computer Science, v. 4063), p. 139–153. 5, 5.3
- [32] SCHMID, H. A.; PFEIFER, M.; SCHNEIDER, T. A middleware-independent model and language for component distribution. In: **SEM '05: Proceedings of the 5th international workshop on Software engineering and middleware**. New York, NY, USA: ACM, 2005. p. 83–89. ISBN 1-59593-204-4. 5, 5.4
- [33] Tuscany. <http://cwiki.apache.org/TUSCANY/>. 5.1
- [34] Fabric3. <http://fabric3.codehaus.org/>. 5.1

## A Apêndice

### Código A.1: Arquivo de implementação do componente LoggingComponent

```
1 /**
2  * Classe de implementação do componente Logging.
3  * <p>
4  *
5  * O componente disponibiliza um método print(), que notifica
6  * todos os objetos "Notify" de uma nova mensagem de log.
7  */
8 @FacetSet({
9     @Facet(name="LoggingFacet", interfaceType=LoggingOperations.class)
10 })
11 public class LoggingComponent implements LoggingOperations {
12
13     /** Descrição do componente */
14     @MetaInfo(type=Type.FULL_DESCRIPTION)
15     public static String DESC;
16
17     /** Objetos remotos a serem notificados dos logs. */
18     @Receptacle(name="NotifyReceptacle")
19     private NotifyOperations[] notifiers;
20
21     /**
22      * Notifica todos os observadores da ocorrência de um novo log
23      * fazendo a distinção do tipo de log pelo texto presente
24      * na mensagem.
25      *
26      * @param msg O texto do log
27      */
28     @Override
29     public void printLogMsg(String msg) {
30         LogType t = null;
31         if (msg.contains("WARN") || msg.contains("warn")) {
32             t = LogType.WARN;
33         } else if ((msg.contains("ERROR") || msg.contains("error"))) {
34             t = LogType.ERROR;
35         } else {
36             t = LogType.OTHER;
37         }
38         printLog(new LogMessageImpl(t, msg));
39     }
40
41     /**
42      * Notifica todos os observadores da ocorrência de um novo log.
43      *
44      * @param msg O objeto contendo o texto e o tipo do log.
45      */
46     @Override
```

```

47 public void printLog(LogMessage msg) {
48     String occursDate = (new Date()).toString();
49     String systemMessage = DESC + " received: " + msg.message;
50
51     System.out.println("[ " + occursDate + " ] " + systemMessage);
52
53     switch(msg.type.value()) {
54         case LogType._WARN :
55             for(NotifyOperations notifier : notifiers) {
56                 notifier.warn(systemMessage, occursDate);
57             }
58             break;
59         case LogType._ERROR :
60             for(NotifyOperations notifier : notifiers) {
61                 notifier.error(systemMessage, occursDate);
62             }
63             break;
64         default :
65             for(NotifyOperations notifier : notifiers) {
66                 notifier.other(systemMessage, occursDate);
67             }
68     }
69
70 }
71
72 public void setNotifiers(NotifyOperations[] notifiers) {
73     this.notifiers = notifiers;
74 }
75
76 public NotifyOperations[] getNotifiers() {
77     return notifiers;
78 }
79
80 }

```

### Código A.2: Arquivo de implementação do componente NotifyComponent

```

1 /**
2  * Classe de implementação do componente NotifyComponent.
3  * <p>
4  *
5  * Disponibiliza os métodos error, other e warn.
6  * Cada método armazena a mensagem do log em um arquivo
7  * específico ao tipo de log ocorrido.
8  */
9 @FacetSet({
10     @Facet(name="NotifyFacet", interfaceType=NotifyOperations.class)
11 })
12 public class NotifyComponent implements NotifyOperations {
13
14     /** Descrição do componente */
15     @MetaInfo(type=Type.FULL_DESCRIPTION)
16     public static String DESC;
17
18     /** Path do arquivo de logs de warnings. */
19     private String warnFilename;
20
21     /** Path do arquivo de logs de erros. */
22     private String errorFilename;
23

```

```

24  /** Manipulador do arquivo de logs de warnings. */
25  private FileWriter warnFile;
26
27  /** Manipulador do arquivo de logs de erros. */
28  private FileWriter errorFile;
29
30  /**
31    * Construtor.
32    */
33  public NotifyComponent() {
34      this.warnFilename = "./tmp/warning";
35      this.errorFilename = "./tmp/error";
36      try {
37          setFilenameSuffix("test");
38      } catch (IOException e) {
39          e.printStackTrace();
40      }
41  }
42
43  /**
44    * Atribui um sufixo aos arquivos de warning e erro.
45    * @param suffix o sufixo
46    * @throws IOException
47    */
48  public void setFilenameSuffix(String suffix) throws IOException {
49      this.warnFilename += "-" + suffix + ".log";
50      this.errorFilename += "-" + suffix + ".log";
51      this.warnFile = new FileWriter(this.warnFilename, true);
52      this.errorFile = new FileWriter(this.errorFilename, true);
53  }
54
55  /**
56    * Atualiza o arquivo de log referência a ocorrência de erros
57    *
58    * @param msg A mensagem do log
59    * @param date A data de ocorrência do log.
60    */
61  @Override
62  public void error(String msg, String date) {
63      String formattedMessage = "[" + date + "] [" + DESC + " :: "
64          + msg + "\n";
65      try {
66          BufferedWriter out = new BufferedWriter(this.errorFile);
67          out.write(formattedMessage);
68          out.flush();
69          out.close();
70      } catch (IOException e) {
71          System.err.println("[INTERNAL ERROR] trying to write to the file "
72              + this.errorFilename + ", the formattedMessage would be: "
73              + formattedMessage);
74          e.printStackTrace();
75      }
76  }
77
78  /**
79    * A implementação atual deste método
80    * atualiza o arquivo de log referência a ocorrência de erros
81    *
82    * @param msg A mensagem do log

```

```

83     * @param date A data de ocorrência do log.
84     */
85     @Override
86     public void other(String msg, String date) {
87         warn(msg, date);
88     }
89
90     /**
91     * Atualiza o arquivo de log referência a ocorrência de warnings
92     *
93     * @param msg A mensagem do log
94     * @param date A data de ocorrência do log.
95     */
96     @Override
97     public void warn(String msg, String date) {
98         String formattedMessage = "[" + date + "] [" + DESC + " :: "
99             + msg + "\n";
100        try {
101            BufferedWriter out = new BufferedWriter(this.warnFile);
102            out.write(formattedMessage);
103            out.flush();
104            out.close();
105        } catch (IOException e) {
106            System.err.println("[INTERNAL ERROR] trying to write to the file "
107                + this.warnFilename + ", the formattedMessage would be: "
108                + formattedMessage);
109            e.printStackTrace();
110        }
111    }
112
113 }

```

---

### Código A.3: Componente MediatorComponent

```

1  /**
2  * Implementação do componente que representa o mediador do chat.
3  * <p>
4  *
5  * O mediador é responsável por receber requisições de um cliente
6  * e repassar aos outros.
7  */
8  @FacetSet({
9      @Facet(name="MediatorFacet", interfaceType=MediatorOperations.class)
10 })
11 public class MediatorComponent implements MediatorOperations {
12
13     /** Descrição do componente */
14     @MetalInfo
15     public static String DESC;
16
17     /** Usuários do chat. */
18     @Receptacle(name="UserReceptacle")
19     private UserOperations[] users;
20
21     /** Gerenciador de locks para operações concorrentes */
22     private ReadWriteLock lockManager;
23
24     /**
25     * Construtor
26     */

```

```
27 public MediatorComponent() {
28     users = new UserOperations[]{};
29     lockManager = new ReentrantReadWriteLock();
30 }
31
32 /**
33  * Método chamado na inicialização do comopnente.
34  */
35 @Lifecycle(eventType=EventType.RUN)
36 public void init() {
37     System.out.println("Chat server - " + DESC + " - running...");
38 }
39
40 /**
41  * Notifica todos os usuários presentes na sala
42  * que um novo usuário se conectou.
43  *
44  * @param newUser o nome do novo usuário
45  */
46 @Override
47 public void sendJoinNotification(String newUser) {
48     lockManager.readLock().lock();
49     for(UserOperations user : users) {
50         if(!user.getName().equals(newUser)) {
51             user.receiveJoinNotification(newUser);
52         }
53     }
54     lockManager.readLock().unlock();
55 }
56
57 /**
58  * Notifica todos os usuários presentes na sala
59  * que um determinado usuário saiu da sala.
60  *
61  * @param leavingUser o nome do usuário
62  */
63 @Override
64 public void sendLeaveNotification(String leavingUser) {
65     lockManager.readLock().lock();
66     for(UserOperations user : users) {
67         if(!user.getName().equals(leavingUser)) {
68             user.receiveLeaveNotification(leavingUser);
69         }
70     }
71     lockManager.readLock().unlock();
72 }
73
74 /**
75  * Repassa para todos os usuários presentes na sala
76  * a ocorrência de uma nova mensagem pública
77  *
78  * @param sender o nome do usuário que enviou a mensagem
79  * @param msg a mensagem
80  */
81 @Override
82 public void sendMessage(String sender, String msg) {
83     lockManager.readLock().lock();
84     for(UserOperations user : users) {
85         if(!user.getName().equals(sender)) {
```



```
86         user.receiveMessage(sender, msg);
87     }
88 }
89     lockManager.readLock().unlock();
90 }
91
92 /**
93  * Envia uma mensagem para um usuário específico
94  *
95  * @param sender o nome do usuário que enviou a mensagem
96  * @param receiver o nome do usuário destinatário
97  * @param msg a mensagem
98  */
99 @Override
100 public void sendWhisper(String sender, String receiver, String msg) {
101     lockManager.readLock().lock();
102     for (UserOperations user : users) {
103         if (user.getName().equals(receiver)) {
104             user.receiveWhisper(sender, msg);
105         }
106     }
107     lockManager.readLock().unlock();
108 }
109
110 /**
111  * Obtém o nome de todos os usuários presentes
112  *
113  * @return o nome de todos os usuários presentes
114  */
115 @Override
116 public String [] getUserNames() {
117     lockManager.readLock().lock();
118     String [] userNames = new String [users.length];
119     int i = 0;
120     for (UserOperations user : users) {
121         userNames[i++] = user.getName();
122     }
123     lockManager.readLock().unlock();
124     return userNames;
125 }
126
127 /**
128  * Obtém os usuários presentes
129  *
130  * @return os usuários presentes
131  */
132 public UserOperations [] getUsers() {
133     return users;
134 }
135
136 /**
137  * Atribui os usuários presentes
138  *
139  * @param users os usuários presentes
140  */
141 public void setUsers(UserOperations [] users) {
142     lockManager.writeLock().lock();
143     this.users = users;
144     lockManager.writeLock().unlock();
145 }
```

```

145 }
146
147 }

```

#### Código A.4: Componente UserComponent

```

1 /**
2  * Componente que representa um usuário do chat.
3  * <p>
4  *
5  * Um usuário recebe notificações de mensagens recebidas
6  * e exibe em sua interface.
7  */
8 @FacetSet({
9   @Facet(name="UserFacet", interfaceType=UserOperations.class)
10 })
11 public class UserComponent implements UserOperations {
12
13   /** Descrição do componente */
14   @MetalInfo
15   public static String DESC;
16
17   /** Nome do componente atribuída pela aplicação */
18   @Property(key="name")
19   private String name;
20
21   /** Mediador do chat. */
22   @Receptacle(name="MediatorReceptacle")
23   private MediatorOperations mediator;
24
25   /** Interface gráfica */
26   private UI ui;
27
28   /** Usuários presentes no chat no momento da conexão deste usuário. */
29   private String [] otherUsers;
30
31   /**
32    * Método chamado quando este componente é carregado no contêiner
33    * @param otherUsers
34    */
35   @Lifecycle(eventType=EventType.LOADED)
36   public void load(String [] otherUsers) {
37     this.otherUsers = otherUsers;
38   }
39
40   /**
41    * Método chamado na inicialização do componente.
42    */
43   @Lifecycle(eventType=EventType.RUN)
44   public void init() {
45     ui = new UI("ASCS Chat – User: " + name, otherUsers);
46     ui.show();
47     ui.setSendAction(new SendAction(ui, this));
48     ui.setCloseAction(new CloseAction(this));
49     ui.displayJoinNotification(name);
50     System.out.println("Chat client (User: " + name + ") – " + DESC
51       + " – running...");
52   }
53
54   @Override

```

```

55  public void receiveJoinNotification(String newUser) {
56      ui.displayJoinNotification(newUser);
57  }
58
59  @Override
60  public void receiveLeaveNotification(String leavingUser) {
61      ui.displayLeaveNotification(leavingUser);
62  }
63
64  @Override
65  public void receiveMessage(String sender, String msg) {
66      ui.displayMessage(sender, msg);
67  }
68
69  @Override
70  public void receiveWhisper(String sender, String msg) {
71      ui.displayWhisper(sender, name, msg);
72  }
73
74  @Override
75  public String getName() {
76      return name;
77  }
78
79  public void setName(String name) {
80      this.name = name;
81  }
82
83  public void setMediator(MediatorOperations mediator) {
84      this.mediator = mediator;
85  }
86
87  public MediatorOperations getMediator() {
88      return mediator;
89  }
90 }

```

### Código A.5: Classe SendAction

```

1  public class SendAction extends AbstractAction {
2
3      /** Interface gráfica */
4      private UI ui;
5
6      /** Componente que representa o usuário do chat. */
7      private UserComponent userComp;
8
9      /**
10      * Construtor
11      * @param ui Interface gráfica
12      * @param userComp Componente que representa o usuário do chat.
13      */
14     public SendAction(UI ui, UserComponent userComp) {
15         super("Send");
16         this.ui = ui;
17         this.userComp = userComp;
18     }
19
20     @Override
21     public void actionPerformed(ActionEvent e) {

```

```
22     String msg = ui.getInputText();
23     if(ui.isWhisperSelected()) {
24         String receiver = ui.getSelectedUser();
25         if(receiver == null || receiver.equals(userComp.getName())) {
26             JOptionPane.showMessageDialog(null,
27                 "Must select a user to whisper. User must be different from
28                 self.",
29                 "Atenção!", JOptionPane.ERROR_MESSAGE);
30             return;
31         }
32         userComp.getMediator().sendWhisper(userComp.getName(), receiver, msg)
33         ;
34         ui.displayWhisper(userComp.getName(), receiver, msg);
35     } else {
36         userComp.getMediator().sendMessage(userComp.getName(), msg);
37         ui.displayMessage(userComp.getName(), msg);
38     }
39     ui.clearInputText();
40 }
```