

Referências Bibliográficas

- [1] ALLCOCK, W.; BRESNAHAN, J.; KETTIMUTHU, R.; LINK, M.; DUMITRESCU, C.; RAICU, I. ; FOSTER, I.. **The Globus Striped GridFTP Framework and Server**. Em: PROC. SUMMER 2005 Super Computing CONF., 2005.
- [2] BENT, J.; VENKATARAMANI, V.; LEROY, N.; ROY, A.; STANLEY, J.; ARPACI-DUSSEAU, A. C.; ARPACI-DUSSEAU, R. H. ; LIVNY, M.. **Flexibility, manageability, and performance in a grid storage appliance**. Em: PROCEEDINGS OF THE 11TH IEEE SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING (HPDC-11), Edinburgh, Scotland, July 2002.
- [3] MADDURI, R.; ALLCOCK, W. ; HOOD, C.. **Reliable file transfers in grid environments**. Em: PROCEEDINGS OF THE THE 27TH IEEE CONFERENCE ON LOCAL COMPUTER NETWORKS, p. 737–738, 2002.
- [4] BENT, J.; THAIN, D.; ARPACI-DUSSEAU, A. ; ARPACI-DUSSEAU, R.. **Explicit control in a batch-aware distributed file system**. Em: PROCEEDINGS OF THE FIRST USENIX/ACM CONFERENCE ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION, March 2004.
- [5] GHEMAWAT, S.; GOBIOFF, H. ; LEUNG, S.-T.. **The google file system**. Em: 19TH ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, Lake George, NY, October 2003.
- [6] FOSTER, I.; KESSELMAN, C.. **Globus: A metacomputing infrastructure toolkit**. IJSA, 11(2):115–128, 1997.
- [7] TANNENBAUM, T.; LITZKOW, M.. **The condor distributed processing system**. Dr. Dobbs Journal, February 1995.

- [8] SANDBERG, R.; GOLDBERG, D.; KLEIMAN, S.; WALSH, D. ; LYON, B.. **Design and implementation of the Sun Network Filesystem**. Em: PROCEEDINGS OF SUMMER 1985 USENIX CONFERENCE, p. 119–130, Portland, OR, 1985.
- [9] HOWARD, J. H.. **An overview of the Andrew File System**. Em: PROCEEDINGS OF WINTER 1988 USENIX CONFERENCE, Dallas, TX, 1988.
- [10] SAMPMANE, G.. **Access Control for Active Spaces**. Tese de Doutorado, University of Illinois at Urbana-Champaign, 2005.
- [11] VISWANATHAN, P.; GILL, B. ; CAMPBELL, R.. **Security architecture in gaia**. Relatório Técnico UIUCDCS-R-2001-2215 UILU-ENG-2001-1720, University of Illinois at Urbana-Champaign, 2001.
- [12] LIMA, M.; MELCOP, T.; CERQUEIRA, R.; CASSINO, C.; SILVESTRE, B.; NERY, M. ; URURAHY, C.. **CSGrid: Um sistema para integração de aplicações em grades computacionais**. Em: ANAIS DO XXIII SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES (SALÃO DE FERRAMENTAS), Fortaleza, Brasil, May 2005. SBC.
- [13] GLASBERG, M. S.; CERQUEIRA, R.. **Activepresentation: A software infrastructure to control presentations in active spaces (in portuguese)**. Em: 10TH BRAZILIAN SYMPOSIUM ON MULTIMEDIA AND THE WEB (WEBMEDIA'04), p. 28–35, Ribeirão Preto, Brazil, 2004.
- [14] GLASBERG, M. S.. **Activepresentation: Um sistema para apresentações distribuídas em ambientes de computação ubíqua**. Dissertação de mestrado, Pontifícia Universidade Católica do Rio de Janeiro, November 2005.
- [15] LIMA, M. J.; SANTOS, M. N.; URURAHY, C.; SILVESTRE, B. O.; MOURA, A. L.; REIS, V. Q.; MELCOP, T.; CERQUEIRA, R. ; CASSINO, C.. **CSBase: A framework for building customized grid environments**. Em: WORKSHOP ON MIDDLEWARE FOR GRID COMPUTING (POSTER SESSION), Grenoble, France, November 2005.
- [16] FOSTER, I.. **Globus Toolkit Version 4: Software for Service-Oriented Systems**. Em: 3779, S.-V. L., editor, PROC. IFIP INTERNATIONAL CONFERENCE ON NETWORK AND PARALLEL COMPUTING, p. 2–13, 2005.

- [17] ALLCOCK, W.; FOSTER, I. ; MADDURI, R.. **Reliable data transport: A critical service for the grid**. Building Service Based Grids Workshop, Global Grid Forum 11, June 2004.
- [18] BESTER, J.; FOSTER, I.; KESSELMAN, C.; TEDESCO, J. ; TUECKE, S.. **GASS: A data movement and access service for wide area computing systems**. Em: PROC. IOPADS'99. ACM, 1999.
- [19] KOSAR, T.; LIVNY, M.. **Stork: Making data placement a first class citizen in the grid**. Em: PROCEEDINGS OF THE 24TH INT. CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, Tokyo, Japan, March 2004.
- [20] BASNEY, J.; LIVNY, M.. **Improving goodput by coscheduling CPU and network capacity**. The International Journal of High Performance Computing Applications, 13(3):220–230, Fall 1999.
- [21] KOLA, G.; LIVNY, M.. **Diskrouter: A flexible infrastructure for high performance large scale data transfers**. Relatório técnico, University of Wisconsin, Computer Sciences Department, 2003.
- [22] ABBAS, A.. **Grid Computing : Practical Guide To Technology & Applications**, chapter 8. Charles River Media, 1a edição, December 2003.
- [23] HATCHER, E.; GOSPODNETIC, O.. **Lucene in Action**. Manning Publications, 2004.
- [24] BOLTON, F.. **Pure Corba**. Sams, 1a edição, 2001.
- [25] VAZHKUDAI, S.; SCHOPF, J.. **Predicting sporadic grid data transfers**. Em: PROCEEDINGS OF 11TH IEEE SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING. IEEE Computer Society, July 2002.
- [26] HOLZNER, S.. **Ant: The Definitive Guide**. O'Reilly Media, Inc., 2nd edição, 2005.
- [27] HITCHENS, R.. **Java NIO**. O'Reilly Media, Inc., 1st edição, 2002.
- [28] IERUSALIMSKY, R.. **Programming in Lua**. lua.org, 2003.
- [29] DAVID K. GIFFORD, PIERRE JOUVELOT, M. A. S.; JR., J. W. O.. **Semantic file systems**. Proceedings of the 13th ACM Symposium on Operating Systems Principles, p. 16–25, October 1991.

- [30] GOPAL, B.; MANBER, U.. **Integrating content-based access mechanisms with hierarchical file systems.** Em: OPERATING SYSTEMS DESIGN AND IMPLEMENTATION, p. 265–278, 1999.
- [31] SECHREST, S.; MCCLENNEN, M.. **Blending hierarchical and attribute-based file naming.** Em: ICDCS, p. 572–580, 1992.
- [32] HESS, C. K.; CAMPBELL, R. H.. **A context-aware data management system for ubiquitous computing applications.** Em: ICDCS '03: PROCEEDINGS OF THE 23RD INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, p. 294, Washington, DC, USA, 2003. IEEE Computer Society.
- [33] HESS, C. K.. **The Design and Implementation of a Context-Aware File System for Ubiquitous Computing Applications.** Tese de Doutorado, University of Illinois at Urbana-Champaign, 2003.
- [34] ROMÁN, M.; HESS, C.; CERQUEIRA, R.; RANGANATHAN, A.; CAMPBELL, R. H. ; NAHRSTEDT, K.. **Gaia: a middleware platform for active spaces.** SIGMOBILE Mob. Comput. Commun. Rev., 6(4):65–67, 2002.

Apêndice A

IDL CORBA

```
1
2 module gridfs {
3
4     module remote {
5
6         /**
7          * FileServer representa o servidor de arquivos remotos e mantem
8          * uma referencia ao diretorio raiz da arvore do sistema de
9          * arquivos exportado. Diversos servidores de arquivo podem
10         * coexistir em uma mesma maquina, utilizando processos e portas
11         * TCP distintas.
12         */
13         interface FileServer;
14
15         /**
16          * RemoteFile representa um arquivo remoto. Um arquivo, como na
17          * maioria dos sistemas de arquivos tradicionais, pode
18          * representar um diretorio ou um arquivo regular.
19          */
20         interface RemoteFile;
21
22         /**
23          * Channel representa o mecanismo de acesso remoto aos arquivos,
24          * definindo fluxos que permitem a leitura e escrita dos dados.
25          */
26         interface Channel;
27
28         /**
29          * ReadChannel representa o canal de leitura de um arquivo
30          * remoto.
31          */
32         interface ReadChannel;
33
34         /**
35          * WriteChannel representa o canal de escrita de um arquivo
```

```

36  * remoto.
37  */
38  interface WriteChannel;
39
40  /**
41  * RandomAccessChannel representa o canal de acesso randomico ao
42  * arquivo remoto. Suas funcionalidades englobam as
43  * funcionalidades dos canais de leitura e escrita.
44  */
45  interface RandomAccessChannel;
46
47  /**
48  * Path eh um caminho representado por uma sequencia de strings ,
49  * todos os elementos da sequencia , exceto o ultimo , representam
50  * um diretorio do sistema de arquivos , ou um mount point
51  * definido para um outro servidor de arquivos. O ultimo elemento
52  * da sequencia pode representar um arquivo regular , um
53  * diretorio , ou um mount point.
54  */
55  typedef sequence<string> Path;
56
57  /**
58  * FileSequence representa uma lista de arquivos , usualmente os
59  * filhos de um diretorio .
60  */
61  typedef sequence<RemoteFile> FileSequence;
62
63  /**
64  * OctetSequence representa uma lista de bytes , usualmente
65  * utilizado na leitura ou escrita de dados , e na obtencao do
66  * hash (MD5) do arquivo.
67  */
68  typedef sequence<octet> OctetSequence;
69
70  /** FieldSequence representa uma lista de strings , usualmente
71  * usadas como chaves para os metadados.
72  */
73  typedef sequence<string> FieldSequence;
74
75  /** Metadata representa uma tupla que pode ser associada aos
76  * arquivos remotos.
77  */
78  struct Metadata {
79      string field;
80      OctetSequence value;
81  };
82

```

```
83  /** MetadataSequence representa uma sequencia de Metadados,
84  *   usada nos arquivos remotos.
85  */
86  typedef sequence<Metadata> MetadataSequence;
87
88
89  /**
90  *   ServerException indica um problema desconhecido ou nao tratado
91  *   pelo servidor.
92  */
93  exception ServerException {
94      string message;
95  };
96
97  /**
98  *   FileAlreadyExistsException indica um conflito de localizacao
99  *   entre um arquivo preexistente e um arquivo sendo criado.
100 */
101 exception FileAlreadyExistsException {
102     Path name;
103 };
104
105 /**
106 *   FileNotFoundException indica que um arquivo nao pode ser
107 *   encontrado.
108 */
109 exception FileNotFoundException {
110     Path name;
111 };
112
113 /**
114 *   FileInUseException indica que um arquivo estah sendo usado
115 *   pelo sistema e que a operacao solicitada nao pode ser
116 *   realizada.
117 */
118 exception FileInUseException {
119     Path name;
120 };
121
122 /**
123 *   NotFileException indica que o arquivo remoto nao corresponde
124 *   a um arquivo regular.
125 */
126 exception NotFileException {
127     Path name;
128 };
129
```

```

130 /**
131  * NotDirectoryException indica que o arquivo nao corresponde
132  * a um diretorio.
133  */
134  exception NotDirectoryException {
135      Path name;
136  };
137
138 /**
139  * NotEmptyException indica que o diretorio nao estah vazio.
140  */
141  exception NotEmptyException {
142      Path name;
143  };
144
145 /**
146  * InvalidPathException indica que um caminho invalido foi
147  * fornecido. Por exemplo:
148  * 1. um caminho fora da arvore exportada.
149  * 2. um caminho contendo o caracter "/" ou "\".
150  * 3. um caminho contendo caracteres invalidos de acordo
151  * com o sistema de arquivos local.
152  */
153  exception InvalidPathException {
154      Path name;
155  };
156
157 /**
158  * ClosedChannelException indica que o canal nao pode ser
159  * utilizado por estar atualmente fechado. Um canal pode
160  * estar fechado por ter recebido uma solicitacao ao seu
161  * metodo close() ou por ter extrapolado o limite de tempo de
162  * inatividade, conforme o mecanismo de leasing. O
163  * fechamento automatico do canal eh necessario para evitar
164  * alocao indefinida de recursos.
165  */
166  exception ClosedChannelException {
167      Channel ch;
168  };
169
170 /**
171  * InvalidStateException indica que o objeto remoto se encontra
172  * em um estado incompativel com a solicitacao atual, como no
173  * caso onde um arquivo foi removido. O atributo message indica
174  * qual o problema enfrentado pelo servidor.
175  */
176  exception InvalidStateException {

```



```

177     string message;
178     Path name;
179 };
180
181 /**
182  * Verificar a declaracao desse elemento no inicio da IDL.
183  */
184     interface RemoteFile {
185
186 /**
187  * Cria um diretorio no caminho especificado. Caso o caminho
188  * possua mais de um elemento, todos os elementos necessarios
189  * sao criados.
190  */
191         RemoteFile createDirectory(in Path name)
192             raises (FileAlreadyExistsException ,
193                 NotDirectoryException , InvalidPathException ,
194                 InvalidStateException , ServerException);
195
196 /**
197  * Cria um arquivo regular no caminho especificado. Caso o
198  * caminho possua mais de um elemento, todos os elementos
199  * necessarios sao criados.
200  */
201         RemoteFile createFile(in Path name)
202             raises (FileAlreadyExistsException ,
203                 NotDirectoryException , InvalidPathException ,
204                 InvalidStateException , ServerException);
205
206 /**
207  * Indica se esse RemoteFile representa um diretorio.
208  */
209         boolean isDirectory()
210             raises (InvalidStateException , ServerException);
211
212 /**
213  * Cria um Mount Point para um outro arquivo, potencialmente em
214  * outro servidor de arquivos. Caso o caminho possua mais de um
215  * elemento, todos os elementos necessarios sao criados e o
216  * ultimo representa o mount propriamente dito.
217  */
218         boolean addMountPoint(in Path name, in RemoteFile target)
219             raises (FileAlreadyExistsException ,
220                 NotDirectoryException , InvalidPathException ,
221                 InvalidStateException , ServerException);
222
223 /**

```

```

224 * Remove um mount point. Como a operação de "remove" requer uma
225 * referencia ao arquivo a ser removido e a tentativa de
226 * recuperar o mount retorna o arquivo alvo do mount, nao eh
227 * possivel remover um mount point atraves do "remove". Essa
228 * funcao remove o mount sem alterar o arquivo alvo.
229 */
230 RemoteFile removeMountPoint(in Path name)
231     raises(NotDirectoryException, InvalidPathException,
232           FileNotFoundException, InvalidStateException,
233           ServerException);
234
235 /**
236 * Recupera o nome do arquivo.
237 */
238 string getName()
239     raises(InvalidStateException, ServerException);
240
241 /**
242 * Recupera toda a sequencia de elementos necessária para
243 * localizar o arquivo em um determinado servidor. Esse caminho
244 * eh absoluto no contexto do servidor de arquivos e nao expoe a
245 * localizacao do arquivo no sistema de arquivos local.
246 */
247 Path getFullName()
248     raises(InvalidStateException, ServerException);
249
250 /**
251 * Copia o conteudo de um arquivo regular sobre outro arquivo,
252 * previamente criado. O metodo de copia indicado eh utilizado
253 * para a transferencia dos dados e deve estar implementado no
254 * servidor. O metodo "CORBA-<BLOCK_SIZE>" eh um dos metodos
255 * disponiveis e sempre pode ser utilizado nas operacoes de
256 * copia. Ele faz uso dos canais de acesso, realizando chamadas
257 * remotas com no maximo BLOCK_SIZE bytes. Por exemplo: o metodo
258 * "CORBA-32768" eh um metodo valido e implementado pelo sistema.
259 */
260 boolean copyTo(in RemoteFile destination, in string method)
261     raises(NotFileException, InvalidStateException,
262           ServerException);
263
264 /** Move um arquivo regular sobre outro arquivo, previamente
265 * criado. Caso a operação seja realizada internamente a um
266 * mesmo servidor de arquivos, o arquivo eh simplesmente
267 * renomeado. Caso dois servidores distintos estejam envolvidos,
268 * uma operacao de copia eh realizada e seguida por uma remocao
269 * do arquivo original.
270 */

```

```

271         boolean moveTo(in RemoteFile destination, in string method)
272             raises(NotFileException, InvalidStateException,
273                 ServerException, FileInUseException);
274
275 /**
276  * Atribui um tamanho arbitrario para um arquivo regular.
277  */
278         boolean truncate(in unsigned long long size)
279             raises(NotFileException, InvalidStateException,
280                 ServerException, FileInUseException);
281
282 /**
283  * Recupera a data de modificacao do arquivo
284  */
285         long long lastModified()
286             raises(InvalidStateException, ServerException);
287
288 /**
289  * Remove um arquivo. Caso o arquivo regular esteja em uso, ou o
290  * diretorio nao esteja vazio, uma excecao eh lancada. A
291  * validacao do uso corrente do arquivo eh feita no contexto do
292  * sistema (sem condiderar o uso por aplicacoes externas). Esse
293  * metodo nao remove uma arvore de diretorios. A semantica e
294  * atomicidade da remocao recursiva sobre uma federacao de
295  * servidores eh fracamente definida e deve ser implementada
296  * externamente, em um determinado contexto de utilizacao.
297  */
298         boolean remove()
299             raises(FileInUseException, NotEmptyException,
300                 InvalidStateException, ServerException);
301
302 /**
303  * Recupera um filho especifico de um diretorio. O caminho pode
304  * conter diretorios e definicoes de Mount Points.
305  */
306         RemoteFile getChild(in Path name)
307             raises(NotDirectoryException, InvalidPathException,
308                 FileNotFoundException, InvalidStateException,
309                 ServerException);
310
311 /**
312  * Recupera a lista de arquivos de um diretorio.
313  */
314         FileSequence getChildren()
315             raises(NotDirectoryException, InvalidStateException,
316                 ServerException);
317

```

```

318  /**
319  * Recupera o arquivo pai. No caso de um Mount Point, retorna o
320  * diretorio pai ao arquivo que estah referenciado no mount (e
321  * nao o diretorio onde o mount foi definido).
322  */
323      RemoteFile getParent()
324          raises(InvalidPathException, InvalidStateException,
325                ServerException);
326
327  /**
328  * Indica o tamanho do arquivo de dados.
329  */
330      unsigned long long size()
331          raises(NotFileException, InvalidStateException,
332                ServerException);
333
334  /**
335  * Calcula o hash (MD5) do arquivo de dados.
336  */
337      OctetSequence hash()
338          raises(NotFileException, InvalidStateException,
339                ServerException);
340
341  /**
342  * Recupera o servidor de arquivos ao qual esse arquivo estah
343  * associado.
344  */
345      FileServer getFileServer() raises(ServerException);
346
347  /**
348  * Recupera o canal de leitura para esse arquivo.
349  */
350      ReadChannel getReadChannel()
351          raises(NotFileException, InvalidStateException,
352                ServerException);
353
354  /**
355  * Recupera o canal de escrita para esse arquivo.
356  */
357      WriteChannel getWriteChannel()
358          raises(NotFileException, InvalidStateException,
359                ServerException);
360
361  /**
362  * Recupera o canal de acesso randomico (leitura e escrita)
363  * para esse arquivo.
364  */

```

```

365     RandomAccessChannel getRandomAccessChannel()
366         raises(NotFileException, InvalidStateException,
367             ServerException);
368
369 /**
370  * Modifica os metadados associados ao arquivo, potencialmente
371  * removendo todas as tuplas caso uma sequencia de tamanho zero
372  * seja passada como parametro. Independente dos metadados
373  * existentes antes da chamada, o novo conjunto de metadados
374  * serah definido exclusivamente pela nova sequencia de
375  * metadados.
376  */
377     void setMetadata(in MetadataSequence metadata)
378         raises(NotFileException, InvalidStateException,
379             ServerException);
380
381 /**
382  * Realiza uma atualizacao da sequencia de metadados armazenados.
383  * Caso uma tupla com uma determinada chave já tenha sido
384  * armazenada previamente, o valor associado será atualizado para
385  * satisfazer o valor presente na nova tupla. Caso nenhuma tupla
386  * tenha sido previamente armazenada com a uma determinada chave,
387  * a nova tupla será armazenada.
388  */
389     void updateMetadata(in MetadataSequence metadata)
390         raises(NotFileException, InvalidStateException,
391             ServerException);
392
393 /**
394  * Recupera toda a sequencia de metadados associada ao arquivo.
395  */
396     MetadataSequence getAllMetadata()
397         raises(NotFileException, InvalidStateException,
398             ServerException);
399
400 /**
401  * Recupera apenas a sequência de metadados associados aos
402  * campos informados como parâmetro.
403  */
404     MetadataSequence getMetadata(in FieldSequence fields)
405         raises(NotFileException, InvalidStateException,
406             ServerException);
407
408 /**
409  * Non-Portable operations
410  */
411

```

```

412  /**
413   * Habilita a permissao de execucao nos sistemas unix.
414   */
415   void enableExecutionPermission()
416       raises(InvalidStateException , ServerException);
417   };
418
419  /**
420   * Verificar a declaracao desse elemento no inicio da IDL.
421   */
422   interface Channel {
423
424   /**
425    * Indica se o canal estah aberto.
426    */
427    boolean isOpen()
428        raises(InvalidStateException , ServerException);
429
430   /**
431    * Recupera o arquivo de dados associado ao canal.
432    */
433    RemoteFile getFile()
434        raises(InvalidStateException , ServerException);
435
436   /**
437    * Fecha o canal.
438    */
439    void close()
440        raises(InvalidStateException , ServerException);
441   };
442
443  /**
444   * Verificar a declaracao desse elemento no inicio da IDL.
445   * Observar que essa interface herda da interface Channel.
446   */
447   interface WriteChannel : Channel {
448
449   /**
450    * Escreve uma sequencia de bytes no canal. O numero de bytes
451    * escritos eh igual ao parametro size , e em caso de sucesso ,
452    * o mesmo valor eh retornado.
453    */
454    long write(in unsigned long size , in OctetSequence buffer)
455        raises(ClosedChannelException , InvalidStateException ,
456              ServerException);
457   };
458

```

```

459  /**
460   * Verificar a declaracao desse elemento no inicio da IDL.
461   * Observar que essa interface herda da interface Channel.
462   */
463   interface ReadChannel : Channel {
464
465   /**
466   * Ignora os proximos bytes do arquivo. O parametro informado
467   * deve ser um valor nao negativo. O numero de bytes efetivamente
468   * ignorados eh retornado.
469   */
470       unsigned long long skip(in unsigned long long num)
471           raises(ClosedChannelException, InvalidStateException,
472                 ServerException);
473
474   /**
475   * Le uma sequencia de bytes do canal. O numero de bytes
476   * lido eh retornado pelo e pode ser no maximo igual ao
477   * parametro informado.
478   */
479       unsigned long read(in unsigned long size,
480                          inout OctetSequence buffer)
481           raises(ClosedChannelException, InvalidStateException,
482                 ServerException);
483   };
484
485   /**
486   * Verificar a declaracao desse elemento no inicio da IDL.
487   * Observar que essa interface herda das interfaces ReadChannel e
488   * WriteChannel.
489   */
490   interface RandomAccessChannel : ReadChannel, WriteChannel {
491
492   /**
493   * Ajusta o ponteiro do arquivo para uma determinada posicao. O
494   * valor informado eh tomado como valor absoluto, dentro do arquivo.
495   */
496       void seek(in unsigned long long num)
497           raises(ClosedChannelException, InvalidStateException,
498                 ServerException);
499   };
500
501   /**
502   * Verificar a declaracao desse elemento no inicio da IDL.
503   */
504   interface FileServer {
505

```

```

506  /**
507   * Recupera o nome simbolico desse servidor de arquivos.
508   */
509   string getName() raises (ServerException);
510
511  /**
512   * Recupera a referencia para o diretorio que corresponde a raiz
513   * da arvore exportada por esse servidor.
514   */
515   RemoteFile getRoot() raises (ServerException);
516
517  /**
518   * Recupera a taxa media de transferencia entre esse servidor e o
519   * servidor cujo nome foi passado como parametro.
520   */
521   double getAverageTransferRateToHost(in string fileName)
522     raises (ServerException);
523
524  /**
525   * Recupera o espaco livre no sistema de arquivos local ao servidor.
526   */
527   unsigned long long getFreeSpace() raises (ServerException);
528
529  /**
530   * Recupera o endereco do servidor a ser utilizado na copia de
531   * arquivos pelos metodos alternativos.
532   */
533   string getCopyServerAddress(in string method)
534     raises (ServerException);
535
536  /**
537   * Funcoes auxiliares para testes e implementacoes nao
538   * documentadas.
539   */
540   WriteChannel getNullWriteChannel()
541     raises (ServerException);
542
543   long long probe (in FileServer target ,
544                   in boolean accessDisk , in long dataSize)
545     raises (ServerException);
546
547   string getMemoryStatus(in boolean callGC)
548     raises (ServerException);
549
550   long long getUsedMemory() raises (ServerException);
551
552   void setProperty (in string key, in string value)

```



```
553         raises (ServerException);
554
555     void shutdown() raises (ServerException);
556 };
557 };
558 };
```

Listagem A.1: IDL