

# Hierarquia de Classes para Suporte da Compressão de Dados

Artur Ferreira Nuno Pereira Paulo Pereira David Coutinho  
Centro de Cálculo, DEEC, ISEL  
Rua Conselheiro Emídio Navarro nº 1, 1949-014 Lisboa  
Tel.: +351-1-8317070 Fax.: +351-1-8317071  
E-mail: { arturj, npereira, palbp, davidpc }@isel.pt

## RESUMO

O presente artigo descreve uma *framework* para compressão de dados. O ponto de partida para a implementação é o modelo teórico da compressão de dados que representa o processo da codificação de fonte. Para efectuar a modelação de acordo com o domínio do problema, recorre-se ao paradigma da programação orientada por objectos, utilizando a linguagem de programação C++. Os aspectos fundamentais na criação da *framework* são: a transposição dos componentes do modelo teórico para as entidades que a constituem, por forma que esta seja uma infra-estrutura didáctica; a flexibilidade e facilidade de utilização e expansão.

Com a introdução de técnicas de pré-processamento, constata-se que a *framework* permite a criação de aplicações que obtêm, sobre o conjunto normalizado de ficheiros *Calgary Corpus*, taxas de compressão superiores às obtidas por produtos comerciais.

**Palavras chave:** *framework*, programação orientada por objectos, compressão de dados.

## INTRODUÇÃO

A estrutura da *framework* deve reflectir a problemática do domínio em causa. Deste modo, o seu desenvolvimento é precedido do estudo aprofundado desse domínio. Por outro lado, o facto de se tratar de uma estrutura semi-acabada [1] implica a necessidade de algum conhecimento do domínio por parte do utilizador [2]. Estas características são determinantes dado que a utilização da *framework* tem um contexto didáctico.

Existem bibliotecas de compressão de dados [3] que seguem o mesmo modelo teórico, mas não constituem plataformas de desenvolvimento com a flexibilidade e expansibilidade pretendidas.

De modo a enquadrar o leitor com a teoria que serviu de base ao desenvolvimento da *framework*, efectua-se a introdução ao problema da compressão de dados, seguida da descrição da hierarquia, relações entre entidades e identificação dos possíveis pontos de expansão (*hot-spots*) [1], assim como das micro-arquitecturas [4] existentes.

Na representação das entidades da *framework* é utilizada a linguagem de modelação *Unified Modeling Language (UML)* [5].

## COMPRESSÃO DE DADOS

O objectivo da compressão de dados é a representação, com mínima redundância, da informação produzida por determinada fonte. A compressão sem perdas segue o modelo teórico genérico [6] [7] [8] composto por várias entidades que se interligam.

## MODELO TEÓRICO

Este modelo foi proposto em 1981 por Rissanen e Langdon [6] tendo como característica essencial a separação bem definida entre modelação e codificação. Esta proposta baseia-se no modelo de sistema de comunicação com texto reduzido desenvolvido em 1951 por Shannon [9].

Os dados a comprimir, produzidos por determinada fonte segundo uma distribuição de probabilidades, são constituídos por símbolos que pertencem a determinado alfabeto de dimensão finita.

A figura 1 mostra o modelo teórico da compressão de dados, no qual se identificam duas entidades:

- modelo;
- codificador / decodificador.

O modelo detém informação sobre a estatística dos símbolos produzidos pela fonte. O codificador atribui o código correspondente a cada símbolo ou conjunto de símbolos, segundo a informação obtida a partir do modelo.

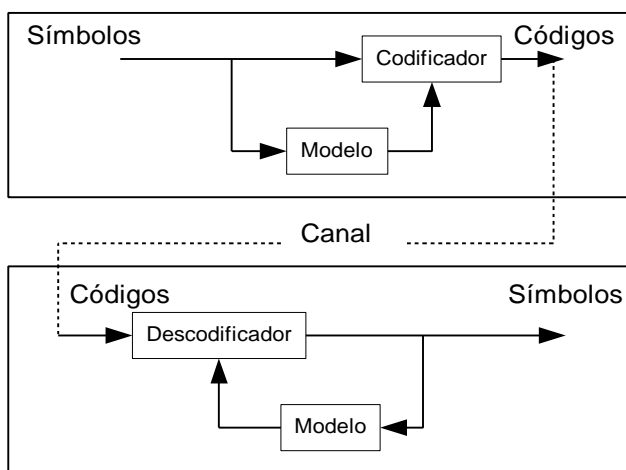


Figura 1 – Modelo da compressão de dados.

Na descodificação efectua-se o processo inverso, isto é, o descodificador recebe os códigos provenientes do canal e converte-os para os símbolos correspondentes em função da informação fornecida pelo modelo.

Deve existir coerência entre os modelos usados na codificação e descodificação, de forma a possibilitar a recuperação do texto original. De acordo com o Teorema da Codificação de Fonte de Shannon [10], a adequação da informação existente no modelo determina a taxa de compressão.

Designam-se por compressor e descompressor os blocos constituídos pelos pares modelo/codificador e modelo/descodificador respectivamente.

## CODIFICAÇÃO E COMPRESSÃO

Existem dois tipos de codificação: estatística e baseada em dicionário. A codificação estatística tem como objectivo atribuir a cada símbolo um código com base na sua frequência de ocorrência, tal que símbolos mais frequentes recebem códigos com menor comprimento que os símbolos menos frequentes. A codificação baseada em dicionário consiste em representar um conjunto de símbolos através de um código que representa a localização desse conjunto de símbolos no dicionário.

Existem três formas de compressão: estática, semi-adaptativa e adaptativa. A sua distinção reside na forma como o compressor constrói o modelo. A compressão estática caracteriza-se pela criação e utilização de um modelo pré-estabelecido. Na semi-adaptativa estabelece-se o modelo em função da amostra a codificar sendo necessário transmitir o modelo para o descompressor, de modo que a descodificação se possa realizar. Nestas duas formas o modelo permanece inalterado ao longo dos processos de compressão e descompressão. Na compressão adaptativa constrói-se um modelo pré-estabelecido que vai sendo actualizado pelo compressor e descompressor ao longo dos processos de compressão e descompressão, respectivamente.

## MODELOS

Existem dois tipos de modelo: estatístico e baseado em dicionário. A informação contida em cada tipo de modelo é diferente, uma vez que está ajustada ao tipo de codificador.

O modelo estatístico é constituído por informação estatística sobre a frequência de ocorrência de todos os símbolos do alfabeto. No caso da codificação baseada em dicionário, o modelo é constituído por uma sequência de símbolos. Neste tipo de codificação a informação está associada a sequências de símbolos e não a cada um dos símbolos do alfabeto. Este tipo de modelo geralmente designa-se por dicionário.

Na codificação estatística o modelo pode ser estabelecido de forma genérica para qualquer codificador. No caso da codificação baseada em dicionário, o conteúdo da informação fornecida pelo modelo ao codificador deve estar adequado às características deste último, causando interdependência entre ambos [8].

## DESCRIÇÃO DA FRAMEWORK

De forma a reflectir as características do domínio do problema, a estruturação interna da *framework* deve seguir todos os aspectos anteriormente apontados relativamente ao modelo teórico.

## MODELOS

Dada a disparidade entre as características dos modelos estatísticos e baseados em dicionário optou-se por efectuar a implementação separada destes dois tipos de modelo.

### ESTATÍSTICO

Os símbolos a codificar pertencem a determinado alfabeto. A entidade *Alphabet* contém o universo de símbolos da fonte, possibilitando a implementação dos compressores e codificadores estatísticos de forma genérica, tal que estes manipulem os símbolos independentemente da sua dimensão e da dimensão do alfabeto.

A entidade *StatisticalModel* agrega o alfabeto em uso e detém informação estatística sobre os símbolos do mesmo, nomeadamente a probabilidade de ocorrência de cada símbolo (estimada a partir da sua frequência de ocorrência) armazenada na forma de função cumulativa de distribuição de probabilidade.

A combinação destas entidades caracteriza completamente a fonte de símbolos pelo que é suficientemente genérica para ser utilizada com qualquer codificador estatístico. A figura 2 ilustra as relações entre as entidades *Alphabet* e *StatisticalModel*.

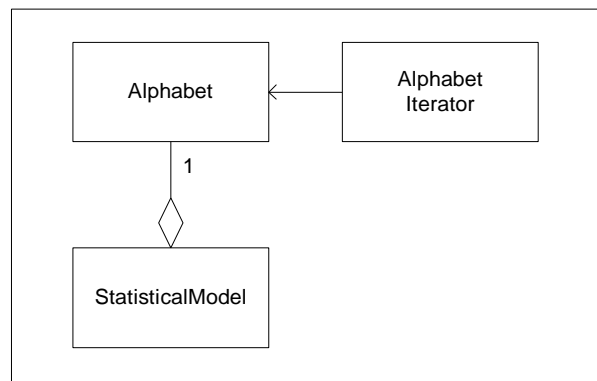


Figura 2 – Micro-arquitetura do Modelo Estatístico.

Para permitir ao modelo a navegação sobre o alfabeto, mantendo a independência entre o primeiro e a implementação do segundo, foi utilizado o padrão de desenho *Iterator* [11].

### DICIONÁRIO

Dadas as características da codificação baseada em dicionário, o alfabeto de símbolos é intrínseco ao modelo. Existe um modelo específico para cada codificador devido à estrita relação entre ambos. O mesmo se verifica para os decodificadores.

Estabeleceram-se as *interfaces* comuns, sob a forma de classes abstractas (*EncoderDictionaryModel* e *DecoderDictionaryModel*) identificadas como ponto de expansão. A figura 3 ilustra a hierarquia de modelos associados à codificação e decodificação, onde se apresentam as entidades *EncoderSlidingWindowModel* e *DecoderSlidingWindowModel* como exemplo de implementações concretas. Estas entidades são compostas por um *buffer* que armazena a sequência de símbolos (dentro de uma janela pré-definida) processados até ao momento. A entidade *SlidingWindowBuffer* é responsável pela implementação do *buffer*. Sobre a mesma foi utilizado o padrão de desenho *Iterator* com os objectivos anteriormente referidos.

### CODIFICADORES

A entidade *Encoder* é a base de todos os codificadores e estabelece a *interface* comum. A hierarquia de codificadores reflecte os dois tipos de codificação através de duas classes abstractas (*StatisticalEncoder* e *DictionaryEncoder*) que estabelecem a relação com o respectivo tipo de modelo.

O processo de codificação tem a seguinte sequência de acções:

- recepção do símbolo ou conjunto de símbolos;

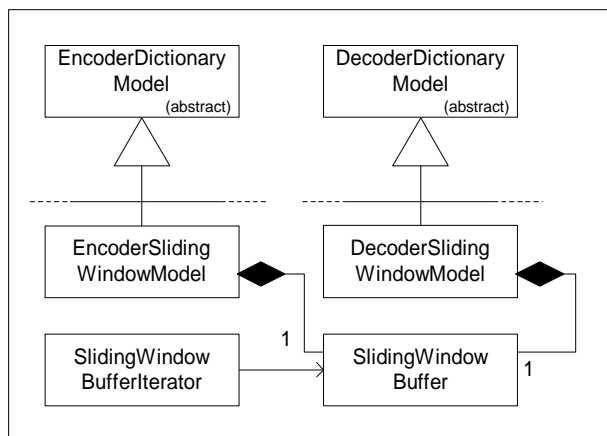


Figura 3 – Micro-arquitetura do Modelo de Dicionário.

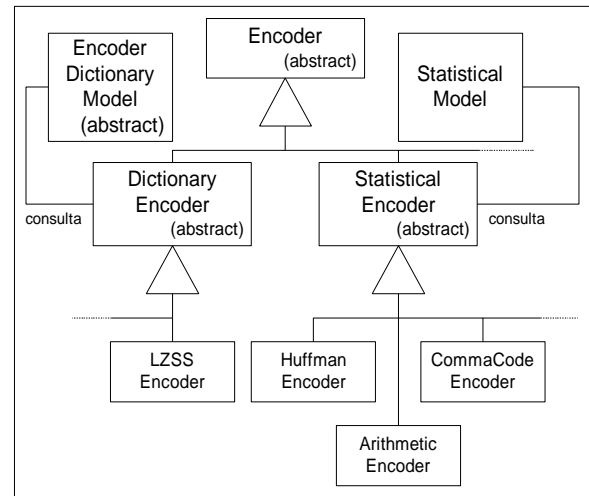


Figura 4 – Micro-arquitetura de codificadores.

- consulta ao modelo;
- atribuição do código.

A figura 4 representa a estrutura hierárquica dos codificadores implementados e relação com os respectivos modelos, salientando os possíveis pontos de expansão.

Os codificadores estatísticos disponibilizados são: *Huffman* [12], Aritmético [13] e *CommaCode*. No que diz respeito aos codificadores baseados em dicionário foi implementada a codificação de *Lempel-Ziv* [14] [15], na variante *LZSS* [16].

A hierarquia dos decodificadores é simétrica à dos codificadores, pelo que não se justifica a sua apresentação.

### TRANSFORMADAS

Para enriquecer a hierarquia está prevista a expansão do modelo teórico da compressão através da introdução de pré-processamento aos dados a comprimir. Este pré-processamento consiste na aplicação de uma ou mais transformações.

A hierarquia que implementa esta funcionalidade está ilustrada na figura 5.

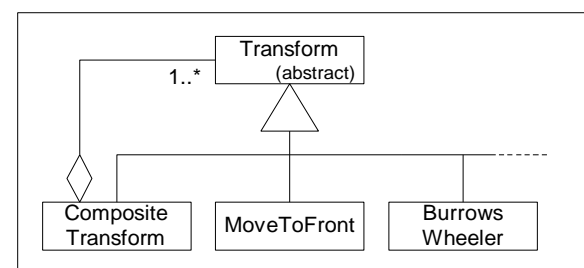


Figura 5 – Micro-arquitetura de transformadas.

As transformações implementadas foram as seguintes:

- transformada de *Burrows-Wheeler* [17];
- transformada *Move-To-Front* [18].

Sobre a hierarquia de transformadas destaca-se a aplicação do padrão de desenho *Composite* [11], que permite sequências de transformadas mantendo o tratamento uniforme, dado que implementa a *interface* definida pela classe base abstracta *Transform*. A entidade *CompositeTransform* é responsável pela implementação da sequência, disponibilizando a *interface* necessária para adicionar e remover transformadas.

## COMPRESSORES

As entidades que efectuam a compressão seguem o conceito geral, tal como documentado na figura 1, onde é sugerido que o compressor utiliza determinado codificador em conjunto com o respectivo modelo. Todos os tipos de compressão referidos anteriormente são suportados pela hierarquia de compressores apresentada na figura 6.

A classe abstracta *Compressor* é a base da hierarquia e para além de definir a *interface* comum a todos os compressores prevê ainda a utilização de transformadas.

A partir desta entidade são estabelecidos os tipos de compressão: estatística, baseada em dicionário e *ad-hoc*. Os dois primeiros são modelados pelas classes abstractas *StatisticalCompressor* e *DictionaryCompressor* que garantem a utilização dos pares codificador / modelo correspondentes ao tipo de compressão usado. O tipo de compressão *ad-hoc* não segue o modelo teórico apresentado. O exemplo mais comum é a codificação de repetições (*Run Length Encoding*) [3], implementada pela entidade *RunLengthCompressor*.

Prevê-se a utilização de sequências de compressores, através da entidade *CompositeCompressor*, recorrendo ao padrão de desenho *Composite*, tal como na micro-arquitectura de transformadas.

A partir das entidades *StatisticalCompressor* e *DictionaryCompressor* efectua-se a especialização para as três formas de compressão existentes, ilustrada na figura 7.

Dada a interdependência entre codificador e modelo, os compressores de dicionário devem assegurar a coerência na relação entre modelo e codificador para cada implementação de compressor.

Dado que o compressor de dicionário está vinculado ao par codificador/modelo que o compõe, este só será definido na altura da criação do compressor em causa. Desta forma justifica-se a necessidade das classes que determinam a forma de compressão serem abstractas.

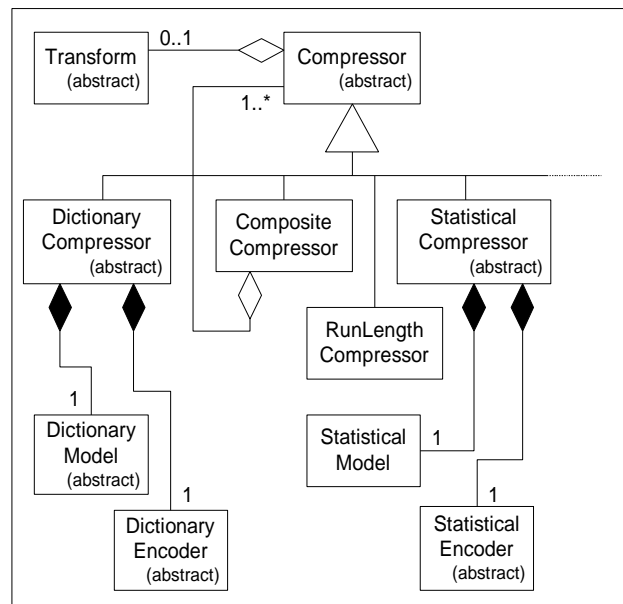


Figura 6 – Tipos de compressor. Relação com modelo e codificador.

De forma análoga à relação entre as hierarquias de codificadores e descodificadores, também as hierarquias de compressores e descompressores são simétricas.

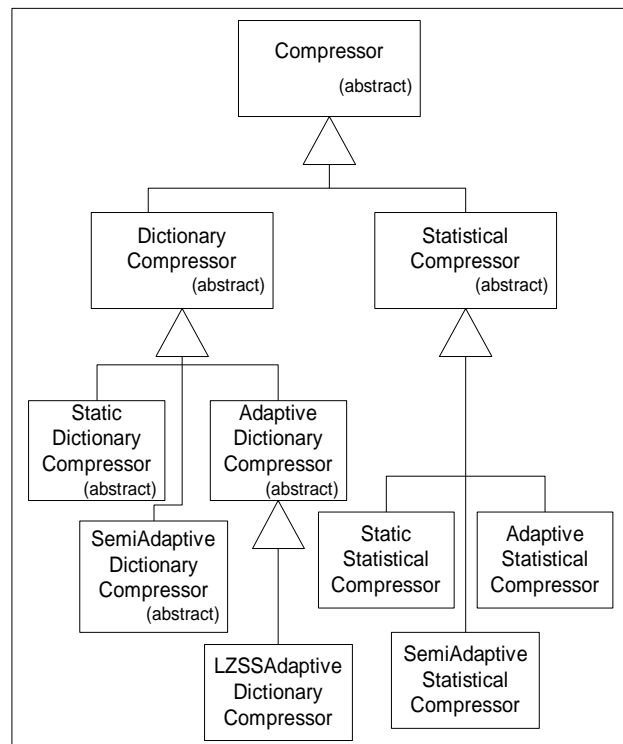


Figura 7 – Formas de compressão.

## ARMAZENAMENTO EM SUPORTE FÍSICO

Tendo em linha de conta que no processo da compressão os símbolos de entrada são transformados em sequências (códigos) com dimensão múltipla do *bit* (dígito binário), foi necessário criar entidades que possibilitem a leitura e escrita dessas sequências. As entidades responsáveis pela manipulação do acesso ao nível do *bit* (*BitIstream* e *BitOstream*) estão representadas na figura 8.

Para além de acrescentar funcionalidade às *streams* da biblioteca *standard* do C++, pretende-se restringir a *interface* disponibilizada pelas mesmas. Para cumprir este objectivo utiliza-se o padrão de desenho *Adapter* [11].

## APLICAÇÃO

Para demonstração das capacidades da *framework*, em termos de taxa de compressão e flexibilidade de utilização, implementou-se o compressor composto pelos componentes indicados na figura 9, e aplica-se ao conjunto normalizado de ficheiros *Calgary Corpus* [19]. O seu desempenho, em termos de taxa de compressão, é comparado com o de produtos comerciais, como por exemplo o PKZIP.

A flexibilidade e simplicidade de utilização da *framework* estão patentes no troço de código correspondente à criação e utilização deste compressor designado por *Comp*.

```
// Criar a transformada composta.
CompositeTransform ct;
BurrowsWheeler bwt;
MoveToFront mtf;
ct.add( &bwt );
ct.add( &mtf );

// Criar os compressores
// Run Length Compressor
RunLengthCompressor rlComp;

// Adaptive Arithmetic Compressor
Alphabet alphabet;
AdaptiveArithmeticEncoderSpeed arithEncoder;
AdaptiveStatisticalCompressor artComp (
    alphabet, &arithEncoder );

// Composite Compressor.
CompositeCompressor comp;
comp.add( &ct );
comp.add( &rlComp );
comp.add( &artComp );

// Efectua a compressão.
comp.compress( "inFile", "outFile" );
```

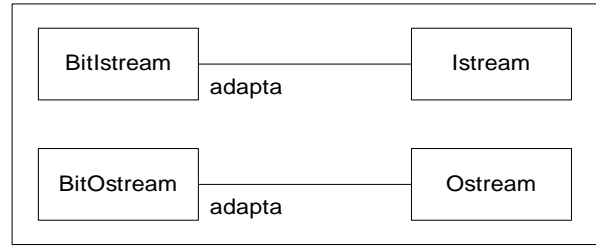


Figura 8 – Entidades para manipulação do acesso ao canal ao nível do dígito binário.

## RESULTADOS OBTIDOS

Na tabela 1 constam os seguintes valores médios:

- dimensão dos ficheiros após compressão;
- percentagem removida [7] [8];
- o número de *bits* para representar cada *byte* [8].

Pela observação dos resultados da tabela 1, conclui-se que o compressor implementado apresenta melhores taxas de compressão que o GZIP e o PKZIP, ficando aquém do BZIP2. A melhoria obtida deve-se essencialmente à combinação da aplicação de transformadas, que efectuem determinada ordenação dos símbolos, seguida da codificação de repetições (*Run Length Encoding*).

Tabela 1 – Resultados obtidos na aplicação de vários compressores sobre o conjunto normalizado de ficheiros *Calgary Corpus*.

	COMP	PKZIP	GZIP	BZIP2
Total	55586	59724	59444	48138
Removed	66%	64%	65%	69%
Bits/Byte	2.74869	2.83500	2.80894	2.48981

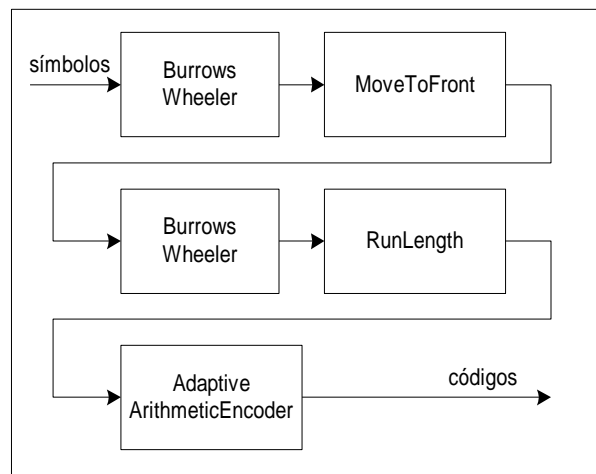


Figura 9 – Diagrama de blocos do compressor *Comp*.

## CONCLUSÕES

No presente artigo apresentou-se o modelo teórico da compressão de dados como ponto de partida para o desenvolvimento de uma *framework* segundo o paradigma da programação orientada por objectos. Conclui-se que este paradigma é indicado para o desenvolvimento de uma plataforma deste género, dado que permite evidenciar a problemática associada ao domínio em causa, satisfazendo o cariz didáctico pretendido.

Sobre o compressor implementado, verifica-se que a aplicação do pré-processamento (transformações) sobre o texto a comprimir é vantajosa, na medida em que aumenta a redundância ao nível do n-grama.

A aplicação de compressores implementados sobre a *framework*, comparativamente aos produtos comerciais apresentados, revela-se mais lenta. Tal verifica-se porque na concepção da mesma os tempos de compressão e descompressão não constituíram critérios de desenho.

Um possível ponto de expansão da *framework* é a criação de uma entidade que suporte alfabeto de dígitos binários. Conjugando a utilização deste alfabeto com compressão aritmética adaptativa conseguem-se elevadas taxas de compressão para ficheiros de imagem [8]. Outros possíveis pontos de expansão são:

- criação de compressores semi-adaptativos que comprimam o modelo antes de o transmitir;
- criação de estruturas de dados optimizadas que acelerem os processos de compressão e descompressão.

## REFERÊNCIAS

[1] W. Pree, “Design Patterns for Object-Oriented Software Development”, Addison-Wesley, 1994.

[2] D. Roberts, R. Johnson, “Evolving Frameworks – A Pattern Language for Developing Object-Oriented Frameworks”, University Of Illinois, <http://st-www.cs.uiuc.edu/users/droberts/evolve.html>

[3] M. Nelson, “The Data Compression Book”, Second Edition, M&T Books, 1996.

[4] R. Lajoie, R. Keller, “Design and Reuse in Object-Oriented Frameworks: Patterns, Contracts, and Motifs in Concert”, Proceedings of the 62<sup>nd</sup> Congress of the Association Canadienne Française pour l’Avancement des Sciences, Canada, May 1994.

[5] M. Fowler, K. Scott, “UML Distilled – Applying The Standard Object Modeling Language”, Addison-Wesley, October 1997.

[6] J. Rissanen, G. Langdon, “Universal Modeling and Coding”, IEEE Transactions on Information Theory, vol. 27, nº1, pp 12-23, January 1981.

[7] R. Williams, “Adaptive Data Compression”, Kluwer Academic Publishers, 1991.

[8] T. Bell, I. Witten, J. Cleary, “Modeling for Text Compression”, ACM Computing Surveys, Vol. 21, nº 4, pp. 557-589, December 1989.

[9] C. Shannon, “Prediction and Entropy of Printed English”, Bell System Technical Journal, Vol. 30, nº 1, pp. 50-64, 1951.

[10] C. Shannon, “A Mathematical Theory of Communication”, Bell System Technical Journal, Vol. 27, pp. 379-423, pp. 623-656, July, October, 1948.

[11] E. Gamma, R. Helm, R. Johnson, J. Vlissides, “Design Patterns, Elements of Reusable Object-Oriented Software”, Addison-Wesley, December 1998.

[12] D. Huffman, “A method for the construction of minimum-redundancy codes”, Proceedings of the Institute of Electrical and Radio Engineers, 40, pp. 1098-1101, September 1952.

[13] J. Rissanen, G. Langdon, “Arithmetic Coding”, IBM J. Res. Dev. 23, nº 2, pp 149-162, 1979.

[14] J. Ziv, A. Lempel, “A universal algorithm for sequential data compression”, IEEE Transactions on Information Theory, vol. 23, nº3, pp. 337-343, May 1997.

[15] J. Ziv, A. Lempel, “Compression of individual sequences via variable-rate coding”, IEEE Transactions on Information Theory, vol. 24, nº5, pp. 530-536, September 1998.

[16] J. Storer, T. Szymansky, “Data Compression via textual substitution”, Journal of ACM, vol. 29, nº4, pp. 928-951, October 1982.

[17] M. Burrows, D. Wheeler, “A Block-Sorting Lossless Data Compression Algorithm”, Digital Systems Research Center Report 124, May 1994.

[18] J. Bentley, D. Sleator, R. Tarjan, V. Wei, “A locally adaptive data compression scheme”, Commun. 29, nº 4, pp. 320-330, April 1986.

[19] I. Witten, T. Bell, “The Calgary/Canterbury text compression corpus”, Anonymous ftp from <ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus>