

REFATORAÇÃO DO ARCABOUÇO DE MODELOS PROBABILÍSTICOS TOPS

Renato Cordeiro Ferreira¹ e Alan Mitchell Durham¹

¹ Instituto de Matemática e Estatística - Universidade de São Paulo

Introdução

O ToPS (*Toolkit for Probabilistic Models of Sequences*) é um arcabouço que contém 8 implementações de modelos probabilísticos publicadas [4] e outras em desenvolvimento. É utilizado como base do sistema criador de preditores de genes MYOP [3], usado em experimentos de bioinformática. Neste trabalho, objetivamos refatorar [1] o ToPS, de modo a facilitar sua compreensão, uso e manutenção, propiciando um ambiente mais amigável para futuras extensões.

Motivação

Há uma série de atributos desejáveis que podem ser utilizados para analisar um sistema. A falta dessas qualidades se manifesta na forma de **maus cheiros** [5], que podem ser classificados segundo os princípios que infringem:

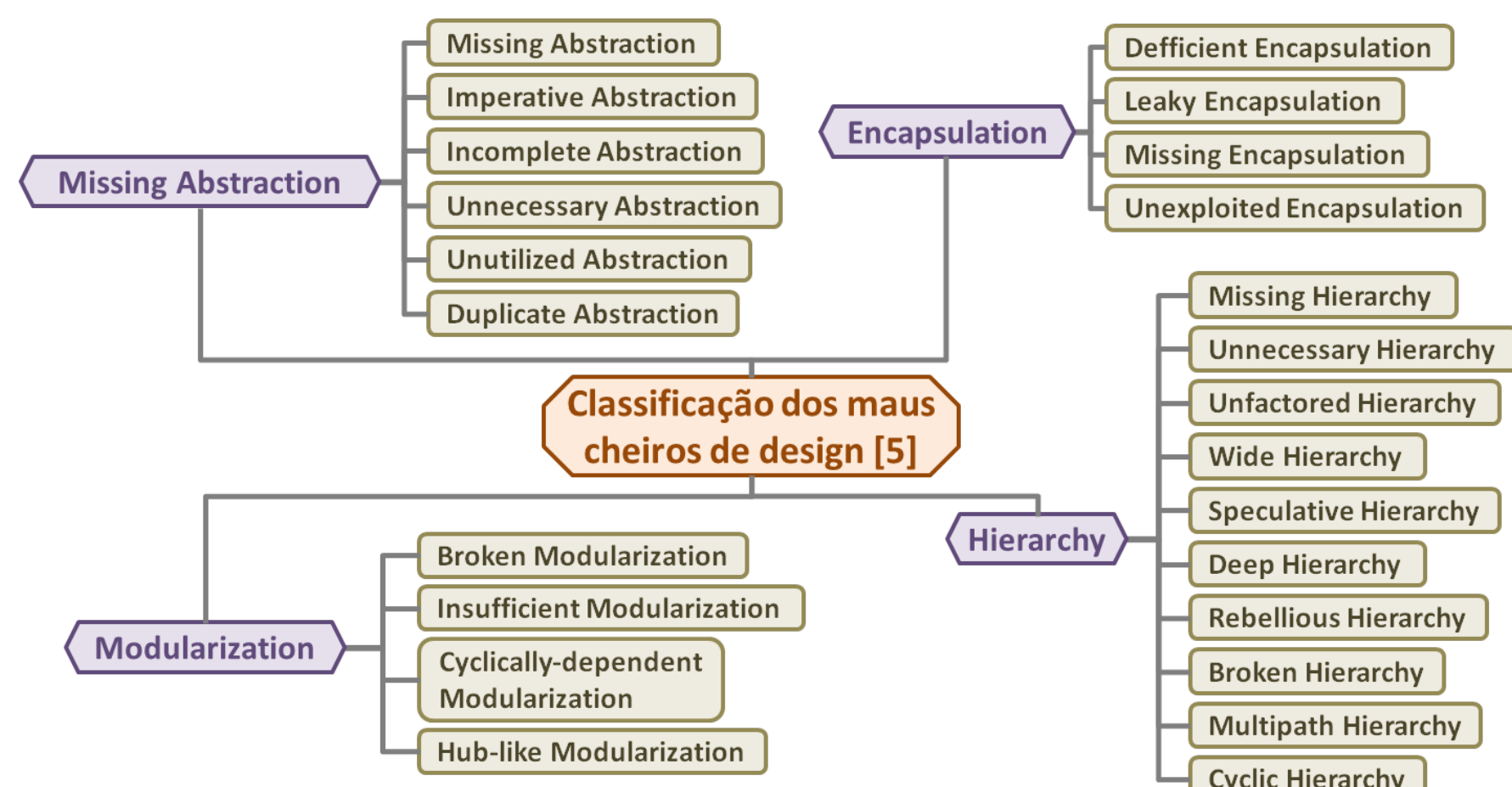


Diagrama 1: Maus cheiros de design.

A arquitetura do ToPS é composta de três hierarquias de classe principais:

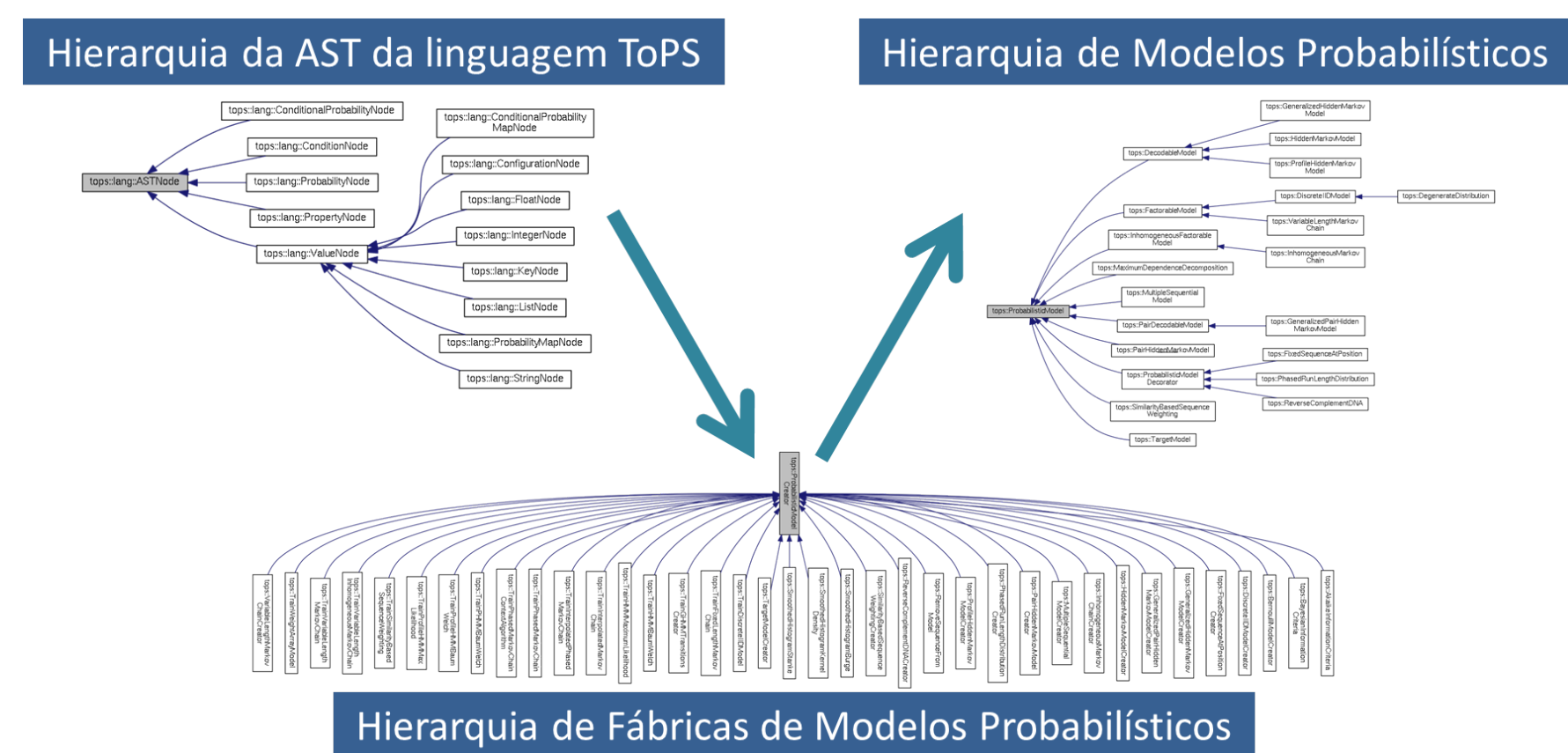


Diagrama 2: Componentes do ToPS e suas dependências.

Analisando as hierarquias relacionadas aos modelos probabilísticos, podemos verificar os seguintes maus cheiros, que servem de motivação para as refatorações propostas neste trabalho:

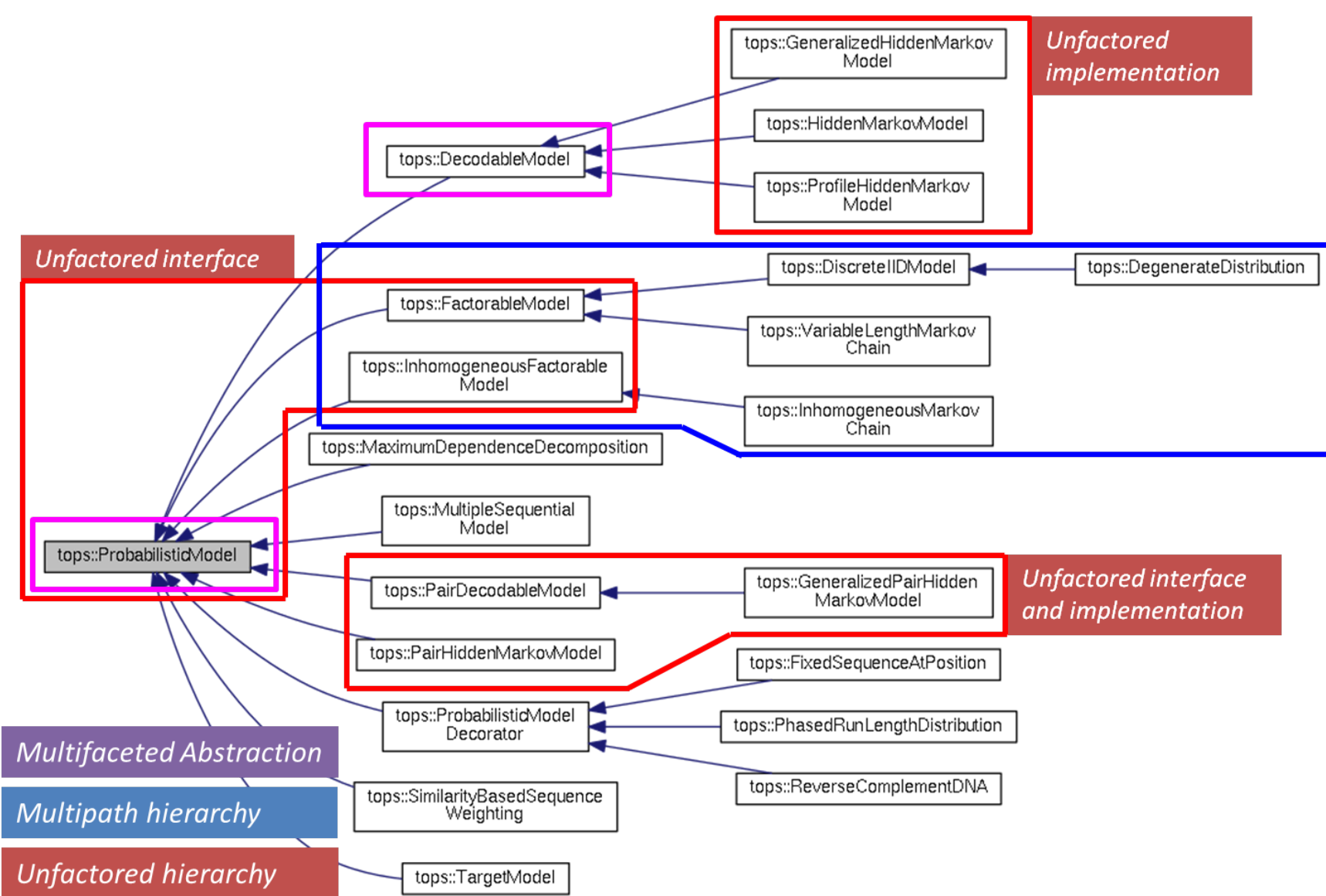


Diagrama 3: Maus cheiros de design na hierarquia de modelos probabilísticos do ToPS.

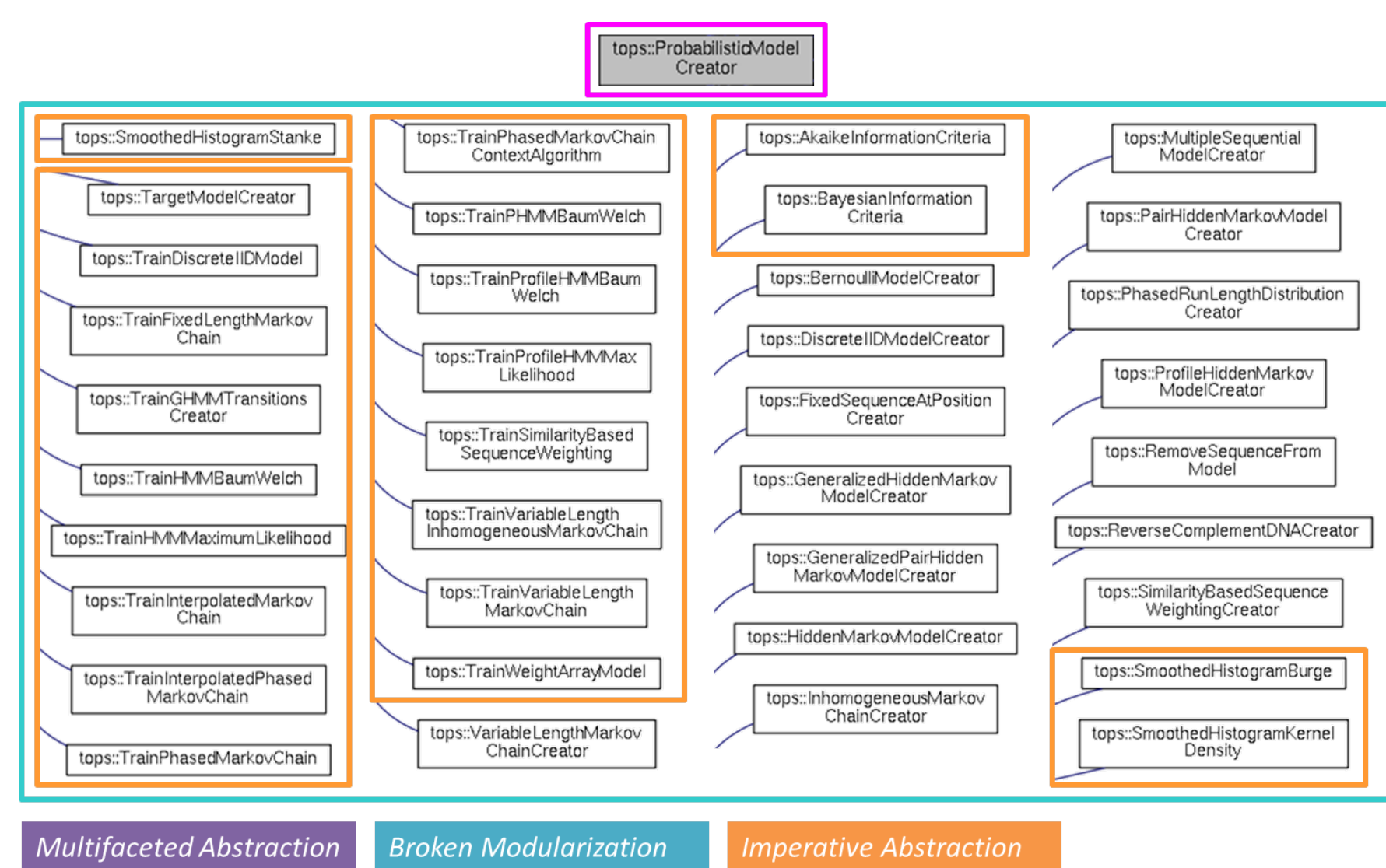


Diagrama 4: Maus cheiros de design na hierarquia de fábricas de modelos probabilísticos do ToPS.

Refatoração

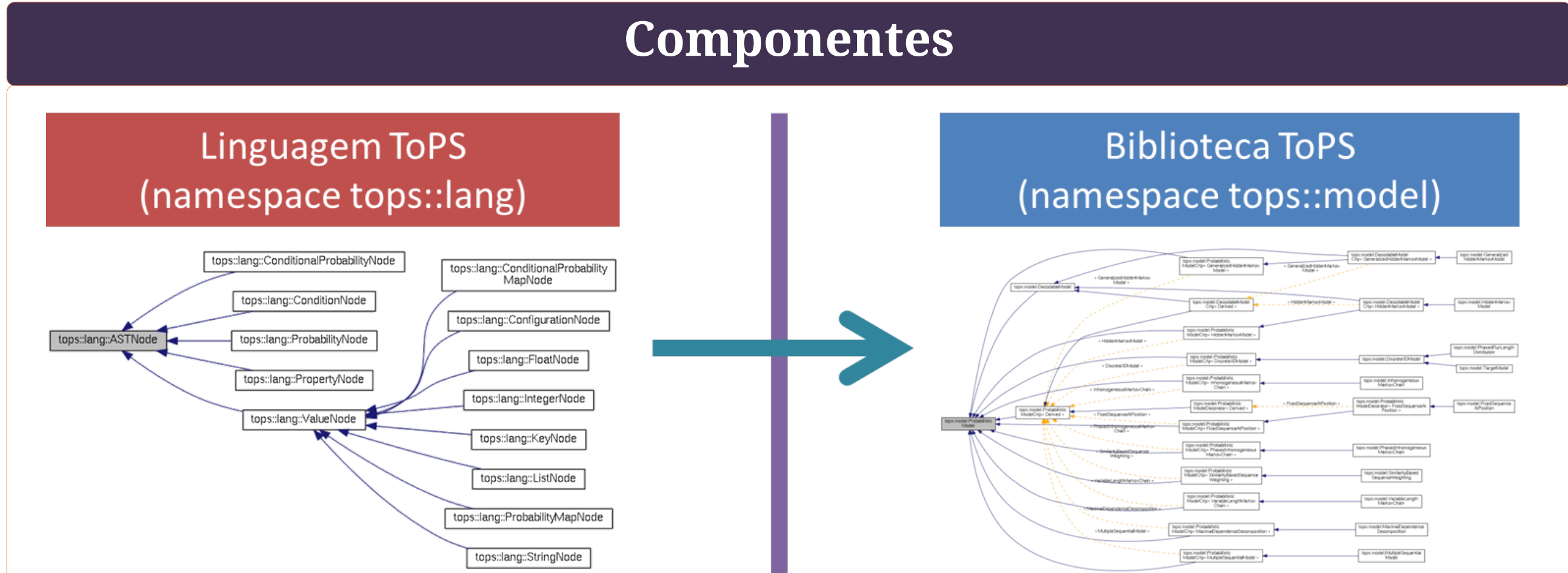


Diagrama 5: Componentes: os algoritmos da hierarquia de fábricas de modelos são redistribuídos. A biblioteca torna-se desacoplada, permitindo usar o ToPS com outras linguagens de programação e especificação.

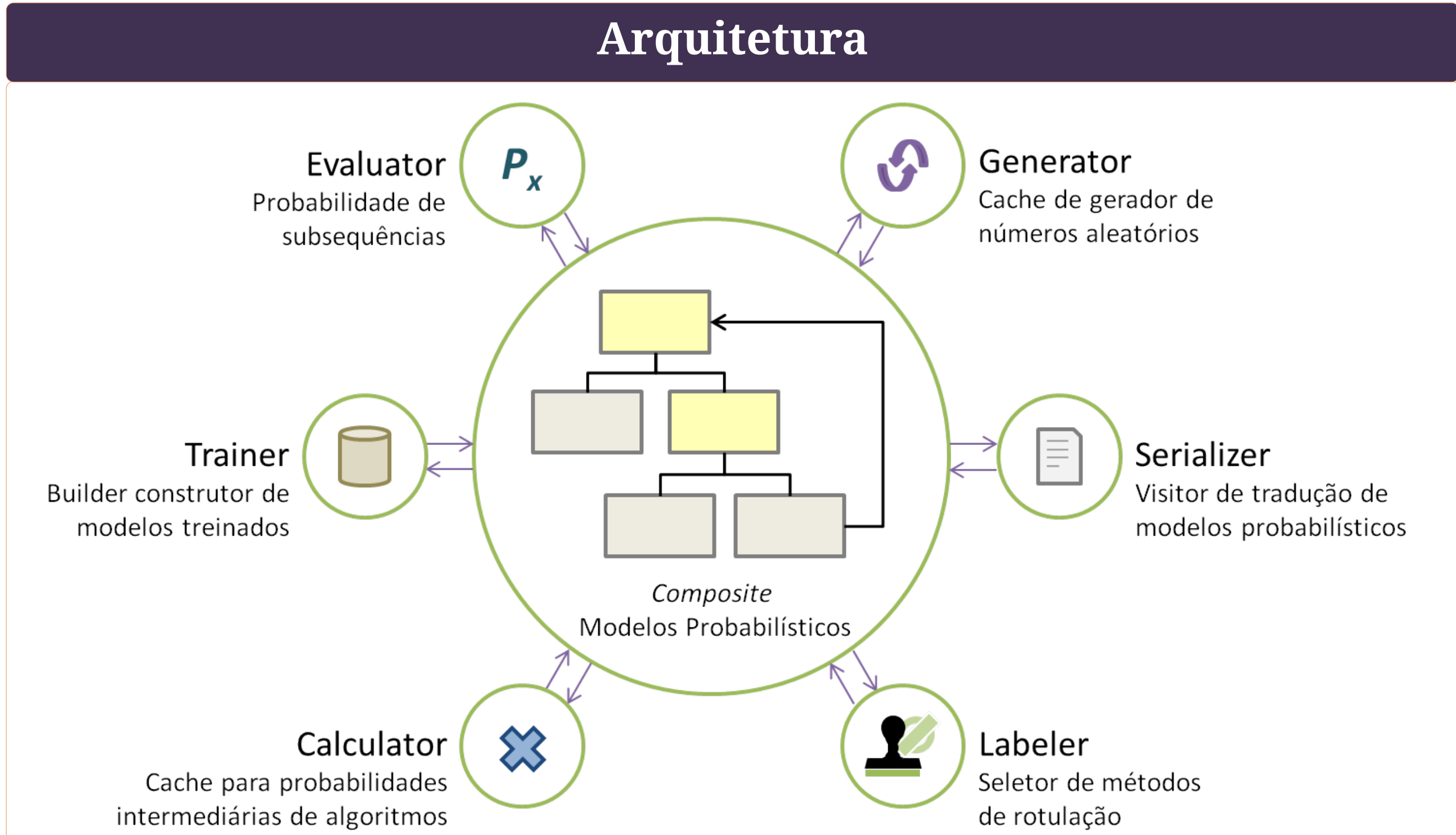


Diagrama 6: Arquitetura de front-ends: o acesso às diferentes funcionalidades da composite de modelos probabilísticos (back-ends) é encapsulada e acessada por meio de classes auxiliares (front-ends).

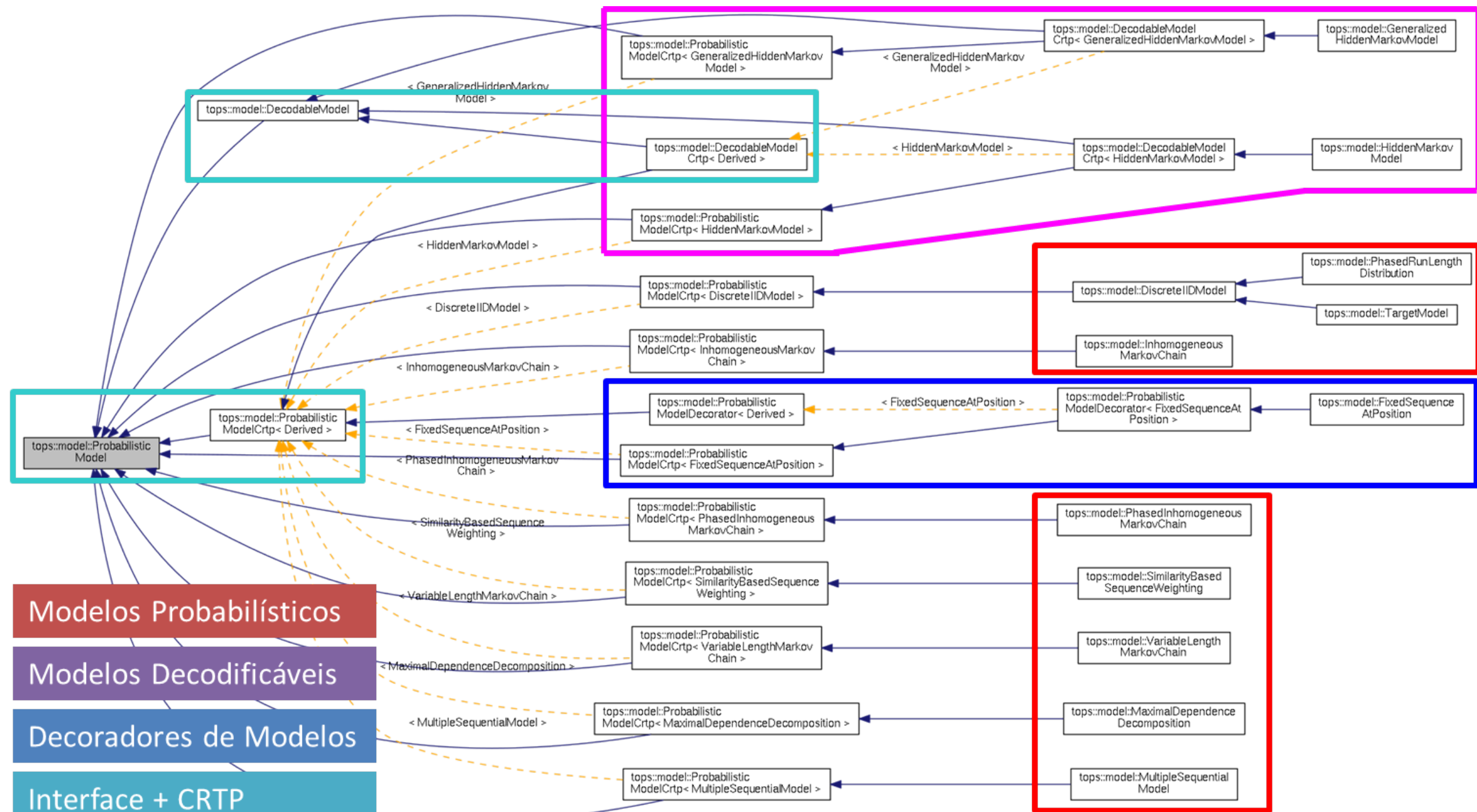


Diagrama 7: Hierarquia de modelos probabilísticos: A versão refatorada contém interfaces para modelos decodificáveis (composite) e modelos decorados (decorator) [2]. O reuso de código é melhorado pelo uso do *Curiously Recurring Template Pattern* (CRTP).

Conclusão

S Single Responsibility Principle (SRP)	Uma classe deve ter um, e apenas um, motivo para mudar
O Open Closed Principle (OCP)	Deve-se ser capaz de estender o comportamento de uma classe, sem modificá-la
L Liskov's Substitution Principle (LSP)	Classes derivadas devem ser substituíveis por suas classes base
I Interface Segregation Principle (ISP)	Muitas interfaces específicas são melhores que uma interface de propósito geral
D Dependency Inversion Principle (DIP)	Deve-se depender de abstrações, não de concretizações

Tabela 2: Princípios da programação orientada a objetos.

Referências

- [1] Martin Fowler et al. "Refactoring: Improving the Design of Existing Code". Em: *Xtemp01* (1999), pp. 1–337. DOI: 10.1007/s10071-009-0219-y.
- [2] Erich Gamma et al. *Design patterns: elements of reusable object-oriented software*. Vol. 47. Addison-Wesley Longman Publishing Co., Inc., 1995, p. 429. DOI: 10.1016/j.artmed.2009.05.004.
- [3] André Yoshiaki Kashiwabara. *MYOP/ToPS/SGEval: A computational framework for gene prediction*. Fev. de 2012. URL: <http://www.teses.usp.br/teses/disponiveis/45/45134/tde-02042012-184145/>.
- [4] André Yoshiaki Kashiwabara et al. "ToPS: A Framework to Manipulate Probabilistic Models of Sequence Data". Em: *PLoS Computational Biology* 9.10 (out. de 2013). Ed. por Hilmar Lapp, e1003234. DOI: 10.1371/journal.pcbi.1003234. URL: <http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003234>.
- [5] Girish Suryanarayana, Ganesh Samarthyam e Tushar Sharma. *Refactoring for Software Design Smells: Managing Technical Debt*. Morgan Kaufmann Publishers Inc., nov. de 2014, p. 244. URL: <http://dl.acm.org/citation.cfm?id=2755629>.

Para mais informações sobre o trabalho, consulte <http://renatocf.github.io/MAC0499/> ou envie um e-mail para renatocf@ime.usp.br

Repositório



Figura 1: Desenvolvimento: *issue tracking* (Waffle), execução e coleta de cobertura de testes (Travis, Coveralls) no GitHub.

Código

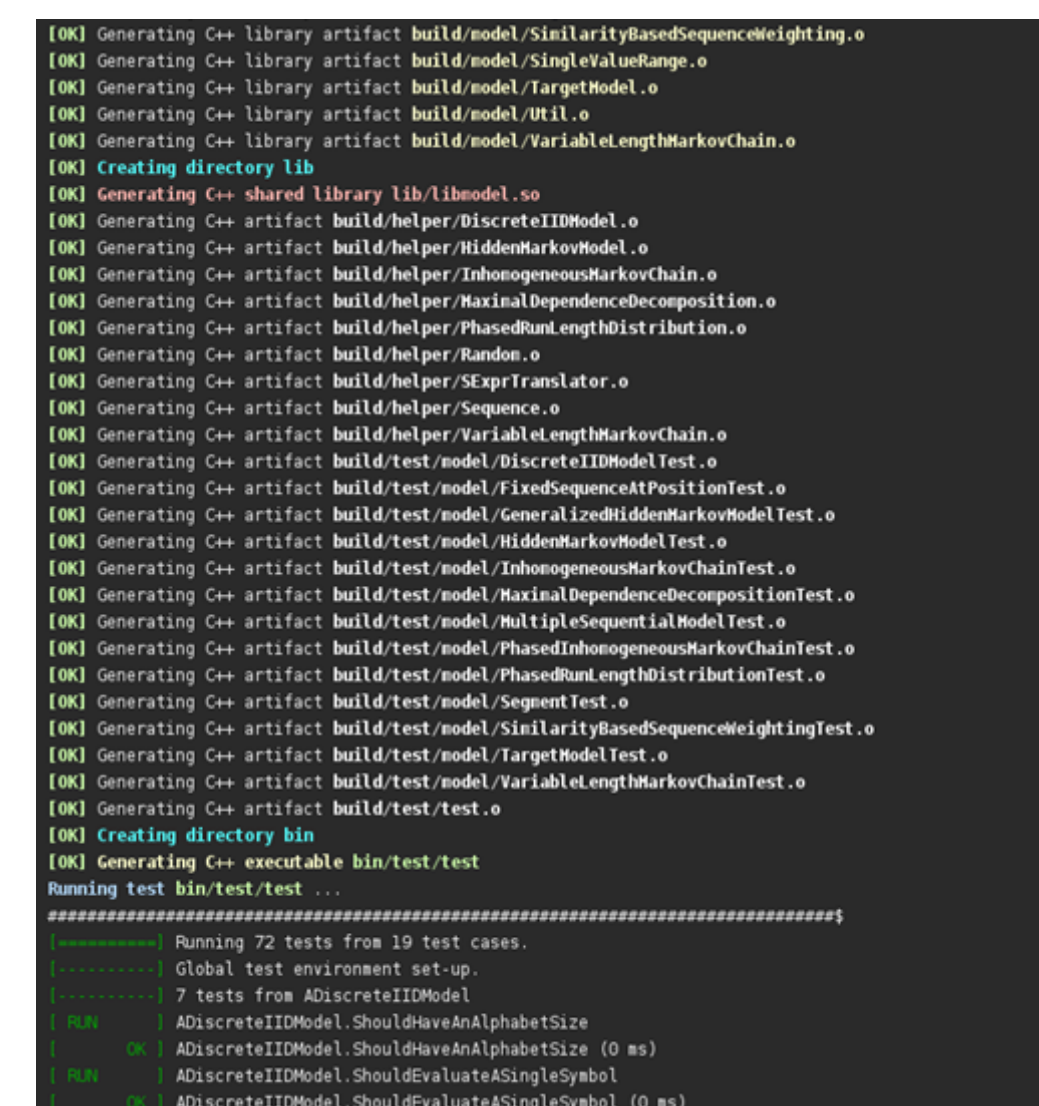


Figura 2: Automatização: o AIO Makefile foi utilizado para compilação, execução de testes e aplicação de ferramentas de análise de código.



Figura 3: Compilação: o código foi mudado para ser compatível com nível máximo de warnings de C++14 no gcc e clang.

Análise	Ferramenta / Biblioteca
Estilo de código	cpplint (Google C++ Style Guide)
Verificação estática	cppcheck
Teste de unidade	GTest / GMock
Cobertura testes	lcov
Profiling	gprof

Tabela 1: Análise de código: programas e bibliotecas utilizadas para assegurar a qualidade do código do ToPS.