Refatoração do Arcabouço de Modelos Probabilísticos ToPS

Renato Cordeiro Ferreira

Monografia do Trabalho de Conclusão de Curso
apresentado à disciplina
MAC0499
(Trabalho de Formatura Supervisionado)

Supervisor: Prof. Dr. Alan Mitchell Durham

São Paulo, Novembro de 2015

Conteúdo

Capítulo 1

Introdução

Ao analisar um sistema, existe uma série de atributos desejáveis que indicam a sua boa qualidade. Compreensibilidade, mutabilidade, extensibilidade, reusabilidade, testabilidade e confiabilidade sumarizam algumas das principais características que revelam a presença de um bom design de código (Suryanarayana et al., 2014). Entretanto, todo programa não trivial sofre modificações ao longo do tempo. Ao fazer mudanças para cumprir objetivos de curto-prazo, ou criar alterações sem um entendimento completo do design do sistema, o código começa a perder sua estrutura, tornando-se mais difícil e custoso de manter (Fowler e Beck, 1999).

Para combater essa perda de qualidade no código, podemos utilizar a técnica de **refatoração**, que pode ser definida de duas formas ligeiramente diferentes, segundo o contexto de utilização da palavra (Fowler e Beck, 1999):

- **Refatoração** (substantivo):
  Uma mudança feita na estrutura interna de um programa para fazê-lo mais fácil de entender ou mais barato de modificar, sem alterar o seu comportamento observável.

- **Refatorar** (verbo):
  Reestruturar um programa por meio da aplicação de uma série de refatorações, sem alterar o seu comportamento observável.

Segundo a definição apresentada, **refatorar** implica necessariamente na **não modificação** do comportamento do programa. Realizar estas alterações, porém, ainda demanda tempo, que poderia ser utilizado para a implementação de novas funcionalidades. A primeira vista, o tempo gasto com refatorações aparenta ser um desperdício. Entretanto, ao analisar mais profundamente o caráter das mudanças trazidas por essas refatorações, é possível notar uma série de vantagens a longo prazo que minimizam esse custo, dentre as quais podemos citar:

- **Design**
  Conforme descrito no início da seção, modificações contínuas num sistema tendem a prejudicar a sua estrutura. Em geral, isso é causado pela presença de **duplicações** - trechos de código que realizam a mesma tarefa, em partes diferentes do programa. Para fazer uma alteração na funcionalidade implementada por esses fragmentos, torna-se necessário modificar vários lugares, aumentando a chance de introduzir erros. Sem uma boa forma de documentação, também é possível que alguns desses trechos fiquem não sejam mudados, causando inconsistências. Utilizando as técnicas de refatoração, é possível eliminar essas duplicações,

simplificando o design e separando melhor as responsabilidades entre diferentes componentes lógicos do programa.

- **Compreensão**
  Podemos ver a programação como a arte de expressar soluções de problemas para que o computador as execute. (Stroustrup, 2014). De forma geral, porém, o código-fonte de um programa não é utilizado apenas uma vez, sendo modificado por outros desenvolvedores (ou pelo seu próprio criador) para fazer novas implementações ou correções em períodos de até meses depois da criação do código. Nesses casos, os programadores dependem quase exclusivamente do próprio código e da sua documentação para entender como o programa funciona. A refatoração, nesse contexto, tem o papel de tornar o código mais legível, uma vez que o deixa mais estruturado. Programar foca-se, então, em ser um modo de transmitir o significado da solução implementada, e não apenas implementar essa solução (Fowler e Beck, 1999).

- **Erros**
  Ao deixar o código mais estruturado e compreensível, fica mais simples encontrar erros no programa. Indiretamente, portanto, refatorar diminui tanto a quantidade de erros presentes quanto a chance de introduzir novos deles.

- **Velocidade**
  Conforme descrito nos itens anteriores, um design pouco claro, com excesso de duplicações, faz com que modificar um sistema torne-se um trabalho igualmente replicado. A chance mais alta de introduzir novos erros e a dificuldade em encontrar os antigos amplia o tempo gasto com correções. Ao diminuir ambos os problemas, a refatoração pode ser usada como instrumento para acelerar a velocidade de programação. A longo prazo, esses benefícios compensam o tempo investido ao refatorar, invalidando o argumento de que a refatoração desperdiça o tempo que poderia ser gasto com novas implementações.

Tendo em vista as vantagens de realizar uma refatoração, torna-se necessário identificar quais características presentes em um programa evidenciam a necessidade de refatorá-lo. Para tanto, Fowler e Beck definem o conceito de **mau cheiro** (*bad smells*): "estruturas no código-fonte de um programa que sugerem a possibilidade de refatoração. Neste trabalho, utilizaremos o sistema de classificação de maus cheiros proposto por Suryanarayana et al., que define maus cheiros como sendo estruturas que indicam a violação de princípios fundamentais do design, impactando negativamente na qualidade do código. Esse sistema divide os maus cheiros em quatro grupos, segundo os princípios de design que violam: abstração, encapsulamento, modularização e hierarquia (Diagrama 1.1). Como não encontramos uma versão traduzida do livro, optamos por utilizar os nomes não traduzidos dos maus cheiros, mantendo o sistema de nomenclaturas conciso e uniforme como pretendido pelos autores (Suryanarayana et al., 2014).

1.1 ToPS

1.1.1 Arquitetura

1.1.2 Organização e distribuição

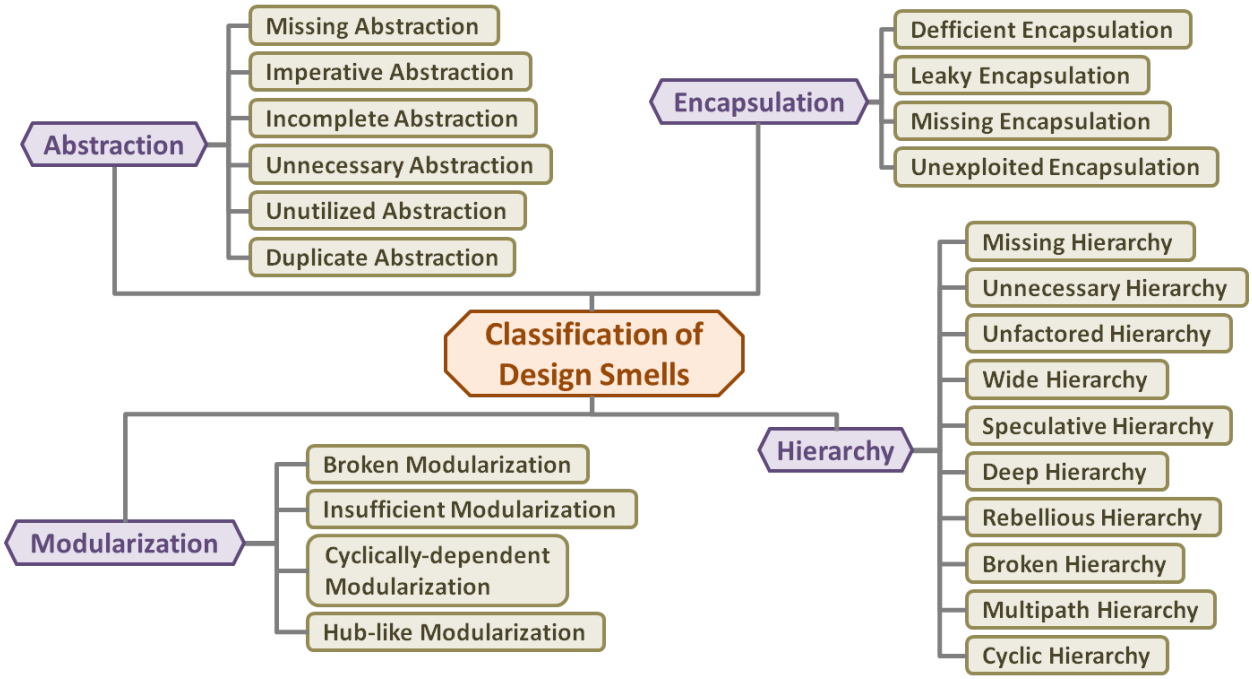

Diagrama 1.1

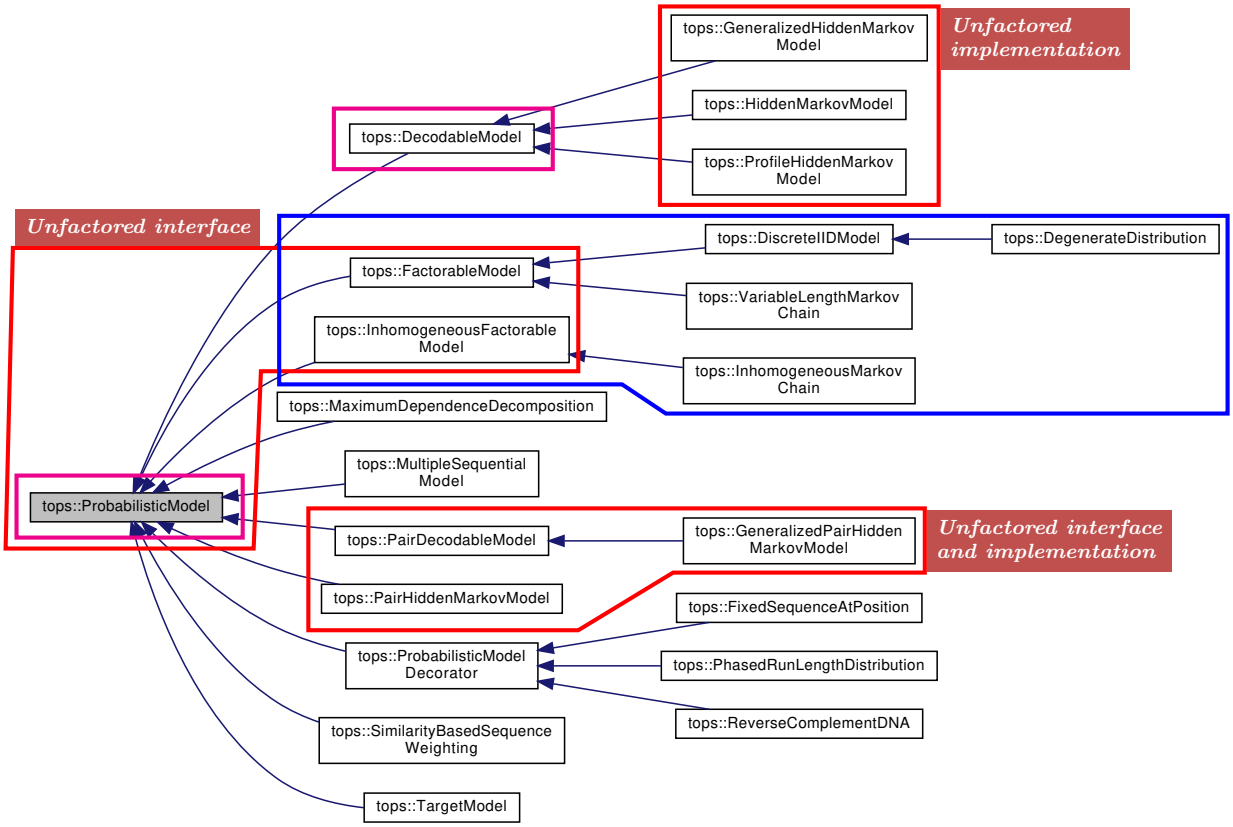

Diagrama 1.2: ***Maus cheiros de design na hierarquia de Modelos Probabilísticos:*** *Analisando os métodos da hierarquia de modelos probabilísticos, identificamos a presença de 3 maus cheiros:* ***Unfactored Hierarchy*** ■ *(expresso em 3 configurações), que indica a presença de duplicações desnecessárias de tipos;* ***Multipath Hierarchy*** ■*, que indica a existência de caminhos desnecessários na hierarquia; e* ***Multifaceted Abstraction*** ■*, que indica que as principais abstrações possuem múltiplas responsabilidades.*

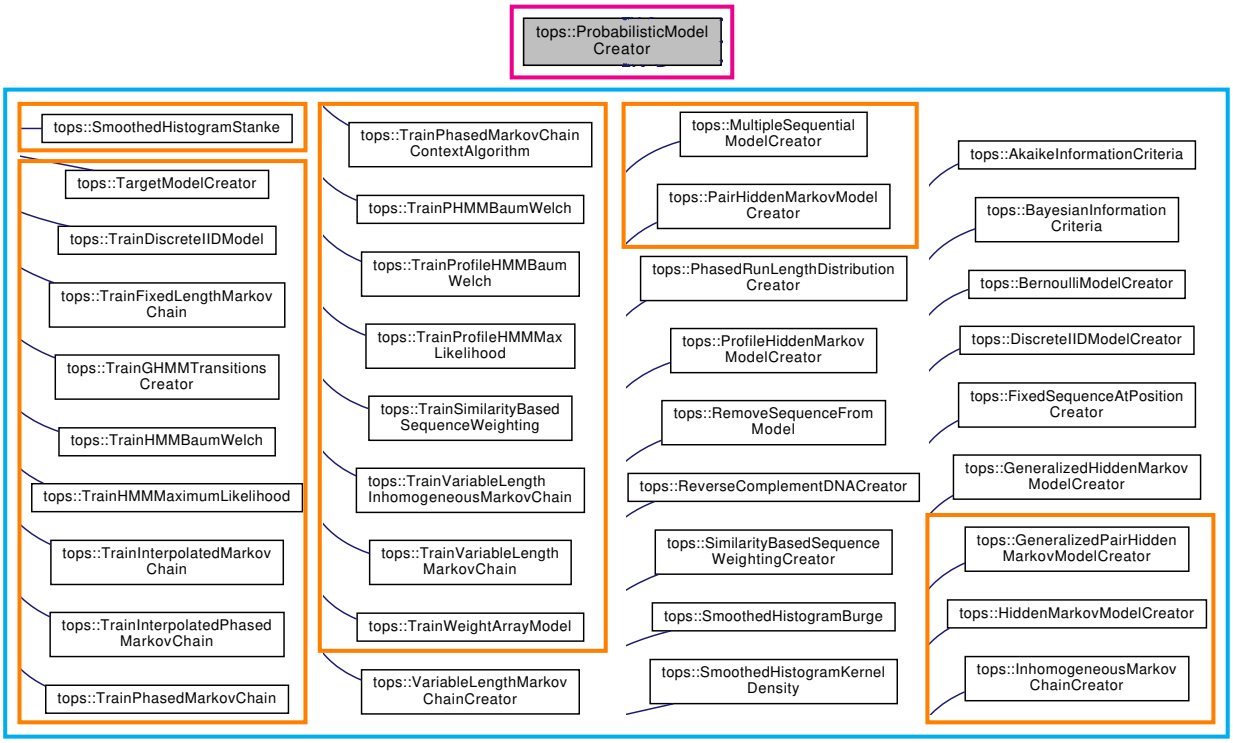

Diagrama 1.3: ***Maus cheiros de design na hierarquia fábricas de Modelos Probabilísticos:*** *As fábricas de modelos probabilísticos apresentam 3 maus cheiros:* ***Multifaceted Abstraction*** ■*, que indica que a abstração da classe possui múltiplas responsabilidades;* ***Broken Modularization*** ■*, que indica a existência de métodos que deveriam pertencer a outras abstrações; e* ***Imperative Abstraction*** ■*, que corresponde a operações que foram transformadas em classes.*

Capítulo 2

Objetivos

Neste trabalho, tínhamos por objetivo refatorar o arcabouço de modelos probabilístico ToPS, realizando modificações em quatro níveis:

- **Código**
  Padronização do estilo de código utilizado no sistema, tendo por base as recomendações do Google C++ Style Guide. Modernização da implementação de classes e algoritmos, removendo a biblioteca boost e utilizando os recursos das novas versões 11 e 14 do C++. Remoção de todos os *warnings* de compilação em seu nível máximo. Compatibilidade total com os compiladores g++ e clang++.

- **Arquitetura**
  Modificação da hierarquia de classes de modelos probabilísticos. Criação de novos métodos e mudança do nome dos antigos para auto-documentação. Separação entre dados utilizados pelos algoritmos e classes que contêm os algoritmos

- **Componentes**
  Separação da linguagem de descrição de modelos e da biblioteca de modelos probabilísticos, tornando a segunda independente da primeira.

- **Repositório**
  Uso de um repositório de armazenamento de código com recursos mais modernos (GitHub). Criação de uma nova política de contribuição para o projeto (baseado em métodos ágeis). Integração com ferramentas de compilação automática e teste. Integração com ferramentas de coleta de métricas de qualidade do código.

Realizamos alterações no arcabouço em todos os níveis, paralelamente. Para começar, entretanto, foi necessário configurar o novo repositório e trazer o código do repositório antigo. Antes de fazer modificações, implementamos testes de unidade para os modelos probabilísticos, de forma a manter os resultados previamente produzidos e evitar a introdução de novos erros (Fowler e Beck, 1999).

No Capítulo 3, descreveremos concretamente os recursos utilizados para realizar a refatoração, segundo os quatro níveis descritos anteriormente.

Capítulo 3

Metodologia

Para explicar os recursos utilizados na refatoração do ToPS, utilizaremos a divisão de tarefas proposta no Capítulo 2. Embora esta separação seja clara, podemos apresentá-la segundo várias abordagens (*bottom-up*, *top-down*, etc). Nas próximas seções, trataremos das alterações em **ordem crescente de complexidade** - que também serve como boa aproximação da ordem cronológica de modificações. Embora algumas mudanças aparentem não ter ligação, as escolhas feitas em um nível de abstração têm consequências nas decisões tomadas nos outros níveis. A ordenação escolhida, entretanto, visa minimizar essas dependências. Deixamos, por fim, para o Capítulo 5 uma discussão mais abrangente das vantagens e desvantagens das escolhas feitas nas próximas seções.

Nesse capítulo, utilizaremos os termos "ToPS 1.0", "ToPS antigo" ou "ToPS original" para nos referirmos à versão não refatorada do ToPS, disponível em http://tops.sourceforge.net/. Usaremos "ToPS 2.0" ou "novo ToPS" como nome da versão refatorada produzida neste trabalho.

3.1 Repositório

O código-fonte do ToPS foi originalmente disponibilizado pelo SourceForge.net - repositório online de código utilizado por vários projetos de software livre de sucesso (como o reprodutor de mídias VLC e a biblioteca Java de mapeamento objeto-relacional Hibernate). O SourceForge aceita projetos que utilizam os sistemas de controle de versão git, mercurial e subversion, e permite que seus usuários criem uma página web estática de forma gratuita, com documentações e informações adicionais sobre o projeto. O SourceForge se destaca na comunidade do software livre por ter sido um dos primeiros a oferecer esse tipo de serviço online.

Entretanto, desde a sua fundação em 2008, o GitHub tem ganhado popularidade dentre projetos feitos com git, tornando-se um dos maiores sites de , servindo .

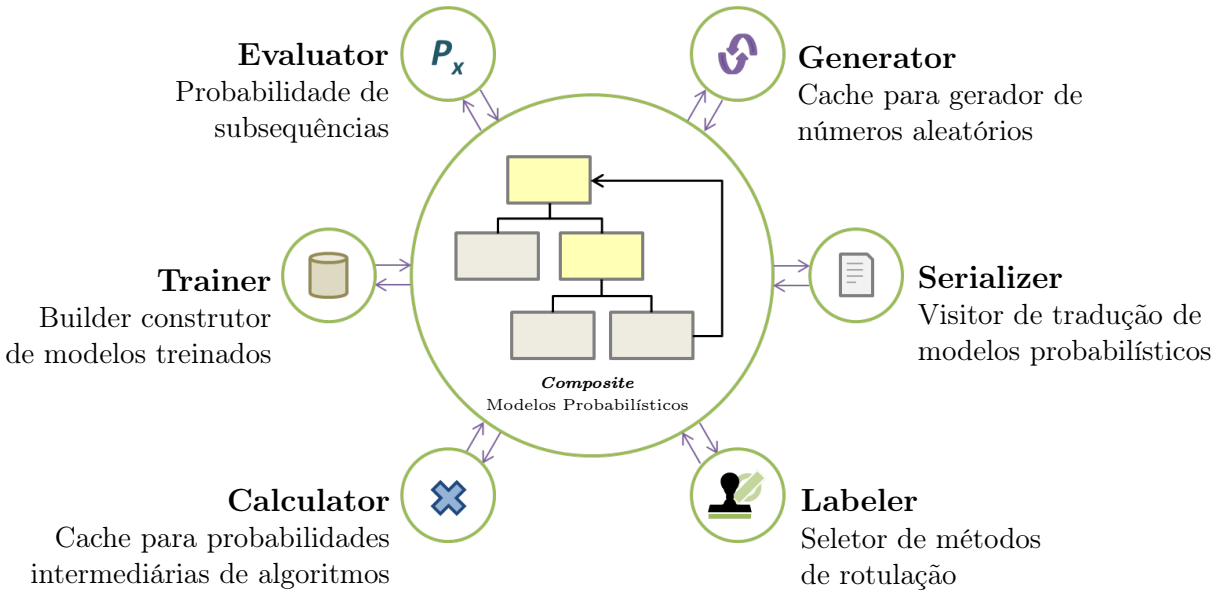

Diagrama 3.1: ***Arquitetura de front-ends****: As funcionalidades da composite (Gamma et al., 1995) de modelos probabilísticos (back-end) são acessadas por meio de classes auxiliares (front-ends), que armazenam dados ou algoritmos utilizados na execução das tarefas.*

3.1.1 Integração Contínua

3.1.2 Contribuição

3.2 Código

3.2.1 Testes de unidade

3.2.2 Análise estática de código

3.3 Componentes

3.3.1 Linguagem (`tops::lang`)

3.3.2 Biblioteca (`tops::model`)

3.4 Arquitetura da biblioteca de Modelos Probabilísticos

3.4.1 Avaliadores

3.4.2 Geradores

3.4.3 Calculadores

3.4.4 Rotuladores

3.4.5 Treinadores

3.4.6 Serializadores

Capítulo 4

Resultados

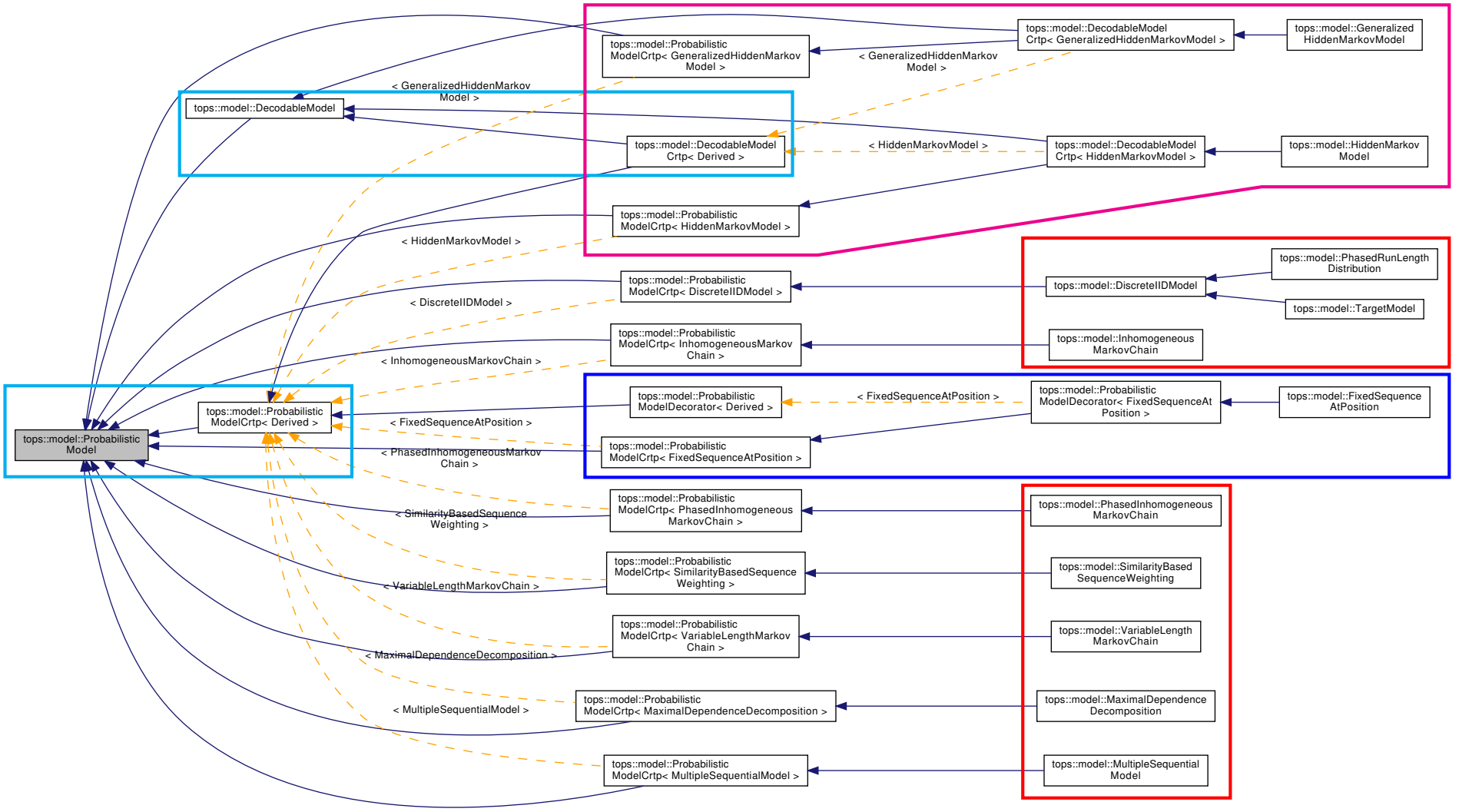

Diagrama 4.1: ***Refatoração da hierarquia de Modelos Probabilísticos:*** *Todos os* ***modelos probabilísticos*** ■ *herdam da classe abstrata* `tops::model::ProbabilisticModel`. ***Modelos decodificáveis*** ■ *e* ***decoradores de modelos*** ■ *possuem suas próprias interfaces, As principais abstrações da hierarquia são implementadas utilizando o* ***Curiously Recurring Template Pattern*** ■ *- idioma do C++ que permite reutilizar a implementação de métodos similares (como os métodos fábrica dos front-ends) por meio de templates.*

Capítulo 5

Análises

Capítulo 6

Conclusões e considerações futuras

Bibliografia

[1] Martin Fowler e Kent Beck. **Refactoring: Improving the Design of Existing Code**. Addison-Wesley, 1999, pp. 1–337.

[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] Bjarne Stroustrup. **Programming: Principles and Practice Using C++ (2nd Edition)**. Addison-Wesley, 2014, p. 1312.

[4] 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.