|  |
| --- |
| Извличане на софтуерни модели с използване на подходящи графи |
|
| Фак. номер: *80867* |
| *Антонио Найденов Николов* |
| *Компютърни Науки, 3ти курс, 1ви поток, 4та група* |
| *antonionikolov93@gmail.com* |
| *Софтуерни технологии, 2015 г.* |

Contents

[1 Въведение 3](#_Toc422874198)

[2 Шаблонен Граф 4](#_Toc422874199)

[2.1 Дизайн Структури и Шаблонен Граф 4](#_Toc422874200)

[2.2 Подробно разглеждане на Шаблонния Граф 5](#_Toc422874201)

[3 Прилагане на шаблони чрез Шаблонните Графи 6](#_Toc422874202)

[3.1 Архитектура на Шаблонно приложния инструмент 7](#_Toc422874203)

[3.2 Генериране на Шаблонен Граф 8](#_Toc422874204)

[3.3 Прилагане на Шаблонните графи: Пример 8](#_Toc422874205)

[4 Заключение 9](#_Toc422874206)

[5 Персонално мнение 10](#_Toc422874207)

[Библиография 10](#_Toc422874208)

Абстрактните периодични промени на голямо количество код често се превръща в необходимост, особено за големите проекти. С цел повишаване на възможността за поддръжка и гъвкавост на такива проекти, шаблонно базираните промени може да изглеждат като добра алтернатива. Ръчната проверка на кода за намиране на места където може да се приложи даден шаблон може да бъде доста времеемка. Затова автоматизирани инструменти могат да помогнат при идентифицирането на места, където могат да бъдат въведени моделите. Нивото на абстракция на изходния код играе важна роля за изграждането на такива инструменти. Ние предлагаме нова абстракция за обектно-ориентиран код, който се нарича "Refactoring Pattern (ReP) Graph" да реализира ефективен шаблонно базиран редакторски инструмент. Графиката REP извлича информацията за кода като по този начин прави процеса по лесен. Предложеният инструмент идентифицира местата на който може да се използват шаблоните за дизайн.

# Въведение

Поддръжката е най-дългата фаза в жизнения цикъл на разработка на софтуер, особено за големи проекти с дълъг живот. Една софтуерна система става кандидат за преработка, когато са възникнали проблеми относно гъвкавостта и поддръжката по време на поправяне на бъгове или добавяне на нова функционалност. В такива случаи шаблонно базираните редакции помага на предприемача да разшири системата с лекота.

Нивото на абстрактност на кода играе важна роля в процеса на откриване на подходящи шаблони. Клас диаграмите предоставят абстрактен поглед на ниво дизайн, но понякога липсва полезна информация която се изисква за идентифицирането на възможни промени. От друга страна кода може да е доста обемист и да има повече от необходимата информация.

Има опити за постигане на шаблонно базирани редакции като “Refactoring to Patterns” която предоставя удобни съвети как да променяме кода си до шаблонно базиран. Но ръчното анализиране на кода е досадна и времеемка задача което ни води до мисълта, че един автоматизиран инструмент за намиране на възможности за редакция може да бъде от голяма полза. Инструмета трябва да може да разпознава места където може да се приложат специфични шаблони като например класове и методи.

# Шаблонен Граф

В този раздел е бегло обяснена дизайн структурната концепция за да направи останалата част от документа по ясна.

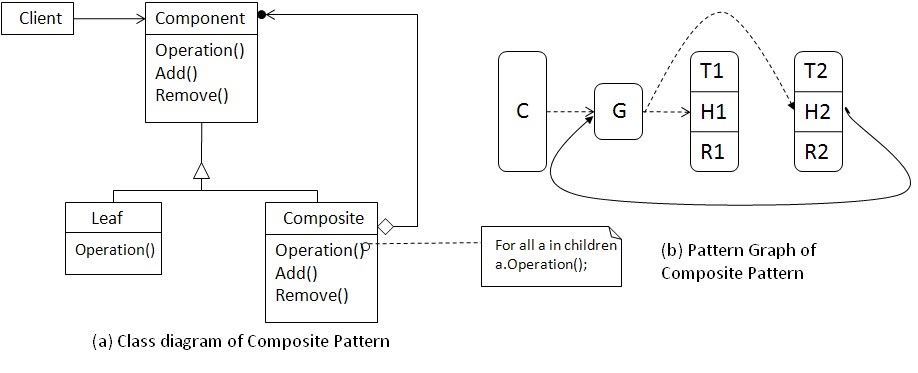
## Дизайн Структури и Шаблонен Граф

Шаблоните за дизайн са общоприети решения на повтарящи се проблеми с дизайна. Също така са дефинирани фундаментални абстракции които се използват за описване на шаблоните за дизайн. Тези абстракции са познати като дизайн структури. Всеки шаблон за дизайн е съставен от една или няколко дизайн структури.

Тази нова абстракция може да се използва за да се види целия дизайн на дадена софтуерна система. Диаграма на дизайна построена от дизайн структури се нарича шаблонен граф. Шаблонния граф е взаимстван от UML клас диаграмата и диаграмата на взаимодействията. Шаблонния граф използва следните означения за да представя информация относно кода. Класовете са представени чрез заоблени правоъгълници и тези правоъгълници имат три компонента. Тези компоненти представян броя на шаблонните методи, hook методите и rigid методите в дадените класове. Hook методите са декларирани в класовете и са дефинирани в подкласовете. Метод който извиква поне един hook метод се нарича шаблонен метод, докато rigid метод е деклариран и дефиниран в един и същи клас.

Множество от класове които се различават само по дефинициите на техните hook методи се третират като единична връзка и се представят само веднъж в шаблонния граф. Клиентски клас е клас който използва шаблона за да изпълни предназначението си. Заоблен правоъгълник без компоненти се използва за да представи клиентска връзка. Връзка едно към едно е представена с прекъсната линия, а връзка едно към много е представена с плътна линия. Входящата връзка показва наследяване или полиморфизъм.

Примерът на **фиг.1** покава означенията на шаблонния граф. На нея е показана клас диаграма и съответния шаблонен граф.



**Фиг. 1**. *Преобразуване на клас диаграма в шаблонна диаграма*

## Подробно разглеждане на Шаблонния Граф

Процеса по преобразуването изисква информация от сорс кода, която не е представена в шаблонния граф. Следователно графа сам по себе си не е полезен относно процеса по промяна на кода. Предложеното решение на Шаблонната редакция е разширение на шаблонния граф. Шаблонния граф представя информацията от сорс кода по абстрактен начин в термините на фундаменталното конструиране. Фундаменталните конструкции са дефинирани от програмните единици като долавят намеренията на шаблоните. Информацията съхранявана в тези конструкции може да бъде достъпена чрез API-то на Шаблонния граф.

| ReP Graph Constructs | Description |
| --- | --- |
| **THR (T, H and R methods)** | Template, hook and rigid method information |
| **Containment, association and create lists** | List of various relationships among classes |
| **Condition, condition-within-condition, condition-within-function** | A condition, a condition within a condition, a condition within a function |
| **Create-within-condition, create-within-function** | A create list within a condition/function |
| **Condition-alt-path-list** | A list of alternate paths for a condition |
| **Updated-var-list-by-condition, updated-var-list-by-function** | Updated variable lists by condition/function |
| **Function-called, passed-params** | Called function list and passed parameters |

**Таб. 1**. *Основени команди на фундаменталните концепции на Шаблонния Граф*

|  |  |
| --- | --- |
| **Design Pattern** | ReP constructs |
| **Abstract Factory** | Create-within-condition |
| **Bridge** | Inheritance-depth, THR |
| **Builder** | Create, Function-called, Passed-params |
| **Composite** | Containment, Function-called |
| **Decorator** | Function-called, THR |
| **Facade** | Association, Function-called |
| **Observer** | Association, Updated-var-list-by-function, Function-called, Passed-params |
| **Singleton** | THR, Member-variables |
| **State** | Condition, Condition-alt-path-list,  Function-called, Updated-var-list-by-condition |
| **Strategy** | Condition, Condition-alt-path-list, Function-called |

**Таб. 2**. *Шаблони и съответните им фундаментални конструкции*

# Прилагане на шаблони чрез Шаблонните Графи

Целта на текущия метод е да преобразува шаблоните от даден сорс код. Метода демонстрира шаблоните на подходящите места в кода за да направи кода по-удобен за поддръжка, по-гъвкав и разбираем. Също така е предложено ново абстрактно ниво на сорс кода, което прави процеса по преработка много по лесен и гъвкав. Шаблонния граф долавя изискванията на кода в термините на фундаменталните конструкции. **Таб. 2** показва няколко шаблона и съответните им фундаментални конструкции използвани за преобразуването им.

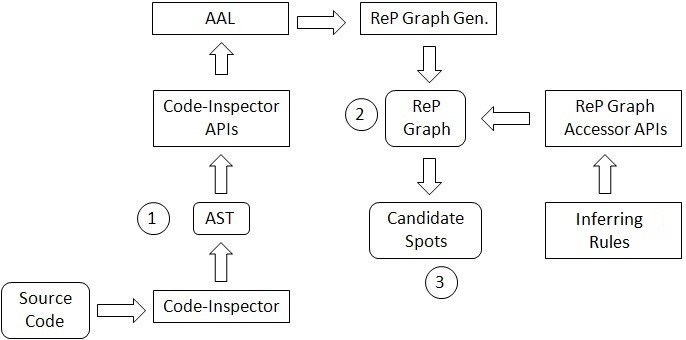
Инструмент на име “Refactor-it” е разработен с цел да реализира тази идея. Инструмента приема сорс код на дадена програма, идентифицира фундаменталните конструкции и ги запазва като шаблонен граф. Инструмента прилага шаблоните като спазва правилата на Шаблонния Граф.

## Архитектура на Шаблонно приложния инструмент

Метода на генериране на шаблонен граф и след това използването му за прилагането на шаблони се състои от три стъпки:

1. Генериране на абстрактно синтактично дърво
2. Генериране на Шаблонен граф
3. Прилагане на шаблоните за дизайн

Както е показано на **фиг. 2**, първата стъпка използва подходящ анализатор за да генерира абстрактното синтактично дърво на дадена софуерна система.



**Фиг. 2**. *Системна архитектура на шаблонно приложния инструмент*

Шаблонният граф може да се достъпи чрез accessor API-то предоставено от шаблонния граф. Правилата за прилагане се записват използваики приложното API на шаблоните. Потребителя може да разширява тези правила като установява правила за допълнителни шаблони.

## Генериране на Шаблонен Граф

Както е показано на **фиг. 2**, генератора използва AAL за да извади необходимата сорс код информация и да генерира шаблонния граф за дадена софтуерна система. RGG използва алгоритъм за да генерира шаблонен граф.

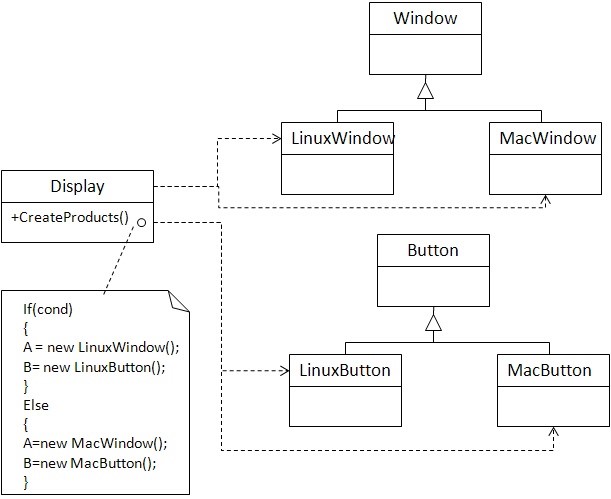
## Прилагане на Шаблонните графи: Пример

**Фиг 3**. показва част от една UML диаграма за примерната програма. Класовете Button и Window заедно с подкласовете им дефинират продуктува иерархия и класа Display създава фамилия от тези обекти като се базира на някакво условие.

Шаблонния инструмент приема тази програма като входни данни и генерира AST с помощта на Code-inspector. Генерирания AST е анализиран и фундаменталните конструкции от сорс кода са идентифицирани. След това инструмента прилага тези конструкции в Шаблонния граф.

Намеренията на шаблона са преведени като правило използвайки съответното API предоставено от Шаблонните Графи. Например намеренията на шаблона abstract factory би трябвало да задоволяват следните клаузи:

* Има условие което има поне две алтернативни възможности.
* Всяка алтернативна възможност трябва да има поне два Create конструктура и броя на тези конструктури трябва да е равен във всяка една от възможностите
* Обектите създадени във всяка алтернативна възможност трябва да имат роднини (алтернативна възможност) във всяка друга алтернатива.



**Фиг. 3**. *UML клас диабрама*

# Заключение

Шаблонния Граф долавя съществената информация във фундаментални конструкции като оставя ненужните дедайли за даден сорс код. Информацията запазена в тези конструкции е достъпна с помощта на accsessor API-то предоставено от Шаблонния Граф. Генерирането на Шаблонния Граф и долавянето на подходящи места за прилагането на шаблони може да се постигне чрез предложената техника. С помощта прдоставените API-та може да се специфицира намеренията на даден шаблон. Нови правила за шаблони могат да бъдат написани лесно като се използват API-тата и може да бъдат добавени като базово правило.

Шаблонния Граф може да бъде използван за измерване на качеството на атрибутите за дадена софтуерна система. Качеството на атрибутите преди и след трансформацията на кода може да бъде сравнено за да се изчисли ефикасността на трансформацията.

# Персонално мнение

Доста полезна информация. До сега не се бях замислял за подобно нещо и го смятам за впечатляващо. Цялата информация е добре поднесена със схеми и таблици които помагат за разбирането на инструмента. Представените алгоритми също доста помагат за възприемането. Автора също така е представил детайно компонентите на Шаблонния Граф което е доста полезно ако искаш да се задълбочиш.

# Библиография

[Ast] Astro. [http://mhuss.com/AstroLib/AstroCpp.zip.](http://mhuss.com/AstroLib/AstroCpp.zip) [Che] Chess. [https://www.pscode.com/vb/scripts/ShowCode.asp?](https://www.pscode.com/vb/scripts/ShowCode.asp?txtCodeId=6337&lngWId=3)

[txtCodeId=6337&lngWId=3.](https://www.pscode.com/vb/scripts/ShowCode.asp?txtCodeId=6337&lngWId=3)

[Cod] Code-inspector. Siemens Corporate Research Princeton, USA.

[Fre] Free framework. [http://www.ebleda.com/opensource/ffw.php.](http://www.ebleda.com/opensource/ffw.php)

[GAA01] Yann-Gaël Guéhéneuc and Hervé Albin-Amiot. Using design patterns and constraints to automate the detection and correction of inter-class design defects. In *TOOLS ’01: Proceedings of the 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39)*, page 296, Washington, DC, USA, 2001. IEEE Computer Society.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns:*

*elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1995.

[HHHL03] Dirk Heuzeroth, Thomas Holl, Gustav Högström, and Welf Löwe. Automatic design pattern detection. In *IWPC ’03: Proceedings of the 11th*

*IEEE International Workshop on Program Comprehension*, page 94, Washington, DC, USA, 2003. IEEE Computer Society.

[JAG+00] D. Janakiram, K. N. Anantharaman, K. N. Guruprasad, M. Sreekanth, S. V. G. K. Raju, and A. Ananda Rao. An approach for pattern oriented software development based on a design handbook. *Ann. Softw. Eng.*, 10(1-4):329–358, 2000.

[JLB02] Sang-Uk Jeon, Joon-Sang Lee, and Doo-Hwan Bae. An automated refactoring approach to design pattern-based program transformations in java programs. In *APSEC ’02: Proceedings of the Ninth Asia-Pacific Software Engineering Conference*, page 337, Washington, DC, USA, 2002. IEEE Computer Society.

[Jne] Jnet library. [http://www.nullsoft.com/free/jnetlib/.](http://www.nullsoft.com/free/jnetlib/)

[K.05] Joshua K. *Refactoring to Patterns*. Addison-Wesley, 2005.

|  |  |
| --- | --- |
| [KJ09] | Viany Kumar Reddy K and D. Jankiram. Design pattern abstraction in c. Technical report, Technical report IITM-CSE-DOS-2006-09. |
| [MS05] | Rajashree MS. Quality estimation model for software development. Technical report, Ph.D. thesis,2005. |
| [Nota] | Notepad. [https://www.pscode.com/vb/scripts/ShowCode.asp?](https://www.pscode.com/vb/scripts/ShowCode.asp?txtCodeId=840&lngWId=3)  [txtCodeId=840&lngWId=3.](https://www.pscode.com/vb/scripts/ShowCode.asp?txtCodeId=840&lngWId=3) |
| [Notb] | Notepad++. [http://notepad-plus.sourceforge.net/.](http://notepad-plus.sourceforge.net/) |
| [ON99] | M. O’Cinnéide and P. Nixon. A methodology for the automated introduction of design patterns. In *ICSM ’99: Proceedings of the IEEE International Conference on Software Maintenance*, page 463, Washington, DC, USA, 1999. IEEE Computer Society. |
| [RJ04] | J. Rajesh and D. Janakiram. Jiad: a tool to infer design patterns in refactoring. In *PPDP ’04: Proceedings of the 6th ACM SIGPLAN international conference on Principles and practice of declarative programming*, pages 227–237, New York, NY, USA, 2004. ACM. |
| [SS03] | Jason McC. Smith and David Stotts. Spqr: Flexible automated design pattern extraction from source code. *Automated Software Engineering,* |

*International Conference on*, 0:215, 2003.