Валидация и оптимизация диаграмм классов UML

М. В. Сергиевский, К. К. Кирпичникова

Национальный ядерный исследовательский университет МИФИ
115409, Москва, Каширское ш., 31

e-mail: *sermax@yandex.ru*

Аннотация. Большинство технологий объектно-ориентированной разработки опирается на использование универсального языка моделирования UML; незаменимыми в процессе проектирования являются диаграммы классов, которые служат для построения модели программной системы. Современные CASE-средства, являющиеся базовыми инструментами объектно-ориентированной разработки, не имеют средств оптимизации UML-диаграмм. В статье будет показано, как можно, базируясь на использовании шаблонов и антишаблонов проектирования, провести верификацию и оптимизацию диаграмм классов. Показано, что некоторые преобразования могут быть осуществлены автоматически; в остальных случаях проектировщику будет указано на потенциальную неэффективность модели и предложены рекомендации по ее исправлению. Предлагается дополнить CASE-средство возможностями верификации и оптимизации диаграмм классов. Для этой цели разработан плагин, анализирующий XMI-файл, содержащий описание диаграмм классов. Поскольку XMI-представление диаграмм классов в Rational Software Architect содержит избыточную информацию, выбрано более оптимальное представление, облегчающее реализацию алгоритмов поиска ошибочных и неоптимальных фрагментов диаграмм классов.
Ключевые слова: UML, диаграмма классов, шаблоны проектирования, антишаблоны, CASE-средство, Rational Software Architect, XMI.

1. Введение

В последнее время именно UML является стандартным средством моделирования объектно-ориентированных программных систем [1]. На этапе проектирования наиболее востребованным инструментом UML являются диаграммы классов (ДК). Известно, что, если на этапе проектирования допущены ошибки, то они распространятся на этап реализации и либо повлекут за собой необходимость дополнительной отладки, либо, в худшем случае, придется прибегнуть к созданию дополнительного прототипа.

Выразительность и широкий выбор конструкций UML отрицательно сказываются на возможностях автоматической верификации — проверки, не содержит ли диаграмма структурных ошибок, несовместимых или избыточных компонентов.

Важно также уметь оценивать степень оптимальности ДК с точки зрения последующей реализации.

Поскольку знания разработчиков в области проектирования ПС обычно недостаточны, приходится сталкиваться с ДК, которые необходимо корректировать. Процесс можно организовать таким образом, чтобы изменения вносились в диаграмму вручную в графическом виде. Но часть изменений можно проводить и автоматически с помощью специализированного CASE-средства, имеющего дополнительные функции валидации. Для этого необходимо уметь анализировать описание ДК и идентифицировать структурные ошибки в диаграммах. Кроме того, желательно иметь набор правил трансформации ДК в соответствии, например, с шаблонами проектирования, которые позволяют оптимизировать модель.

2. Представление ДК

Остановимся на том, какие подходы существуют для представления ДК. Выделим два основных: первый опирается на стандартное описание в виде графа с разными типами вершин и ребер, а, второй — на представление в виде некоей формальной системы. Существу.т несколько способов формального описания ДК; наиболее известны из них те, которые основаны на использовании OCL, языка Z+ и дескрипционных логик. Реальная верификация частично осуществима для двух последних.

Проверка непротиворечивости ДК. Как известно, семантика самого UML не позволяет использовать определенные сочетания конструкций. Например, класс не допускает возможности наследования у самого себя прямо или косвенно, класс-ассоциация не должен иметь те же атрибуты, что и связанные с ним классы и т. д. Формально описать эти ограничения (а их достаточно много) очень сложно. Попытки сделать это приводят к чрезмерному усложнению формальных моделей [10]. Вместо этого можно предложить дополнительные процедуры анализа, которые позволят выявить ошибки такого рода.

Оптимизация ДК, т. е. разработка правил для замены одних конструкций на другие, более оптимальные. Какую ДК считать более оптимальной [7] — непростой вопрос. К примеру, использование многих стандартных шаблонов проектирования [2] приводит к увеличению числа элементов ДК, но улучшает ее качество с точки зрения последующей генерации кода и модифицируемости. Так, ввод интерфейсов в ДК чаще всего повышает универсальность и степень повторной используемости будущего кода, но уменьшает понятность.

Задача оптимизации связана с поиском неэффективных с точки зрения последующей реализации фрагментов ДК. Например, ищутся фрагменты диаграмм, для которых в соответствии с шаблонами проектирования есть более эффективная форма представления. Также может быть применен подход, основанный на поня-

тии антишаблона проектирования — неперспективного для реализации фрагмента ДК [9]. Суть этого подхода заключается в том, что если в ДК обнаруживается антишаблон, то проектировщику дается информация о сути проблемы и предлагается внести изменения в ДК.

3. Примеры оптимизации

Приведем список типичных фрагментов ДК, которые либо содержат ошибки, либо могут быть представлены в более оптимальной форме. Эти фрагменты в ряде случаев описываются в виде шаблонов или антишаблонов проектирования (табл.).

Таблица .

№ п/п	Название фрагмента	Критичность	Реакция
1.	Зацикливание (наследование) для классов	Высокая	Ошибка
2.	Зацикливание (наследование) для интерфейсов	Высокая	Ошибка
3.	Зацикливание (агрегирование)	Высокая	Ошибка
4.	Класс, не связанный с другими элементами ДК	Средняя	Ошибка
5.	Интерфейс, не связанный с классами	Высокая	Ошибка
6.	Множественное наследование классов	Высокая	Предупреждение
7.	Множественное агрегирование	Низкая	Предупреждение
8.	Циклы из отношений ассоциации	Низкая	Предупреждение
9.	Циклы из отношений ассоциации и классов-ассоциаций	Низкая	Предупреждение
10.	Тернарные ассоциации	Низкая	Предупреждение
11.	Шаблон, понижающий арность ассоциации	Низкая	Рекомендация по исправлению
12.	Антишаблон «Клякса»	Средняя	Предупреждение
13.	Шаблон «Цепочка обязанностей»	Низкая	Рекомендация по исправлению
14. .	Одноименные атрибуты у класса и связанного с ним класса-ассоциации	Высокая	Ошибка
15.	Парные отношения зависимости между классами: A ---> B и B ---> A	Низкая	Предупреждение
16.	Ограниченное число методов в классе	Низкая	Рекомендация по исправлению
17.	Ограниченное число параметров метода	Низкая	Предупреждение

Приведем примеры нескольких шаблонов [2, 5, 6] и антишаблонов проектирования, которые используются в данной работе.

Наличие одноименных атрибутов у класса и связанного с ним класса-ассоциации. В принципе одноименные атрибуты в разных классах вполне допустимы, хотя, чтобы избежать двусмысленности, следует стремиться к минимизации их количества. Но дублирование атрибутов у класса и связанного с ним класса-ассоциации свидетельствует о наличии ошибки, поскольку прямо или косвенно ат-

рибут класса-ассоциации относится и к классам, входящим в ассоциацию. В примере на рис. 1 атрибут NAME в классе TIMETABLE является лишним.

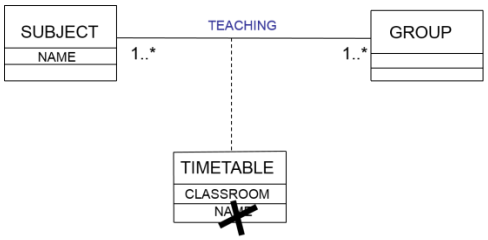

Рисунок 1. Класс-ассоциация с одноименными атрибутами

Шаблон «Цепочка обязанностей». Рассмотрим наиболее простой случай применения этого шаблона (рис. 2), когда запрос Handler() может обрабатываться объектом одного из двух классов.

В этом случае может быть введен абстрактный класс (или интерфейс), перенаправляющий запрос конкретному классу. Тогда ДК примет следующий вид (рис. 3).

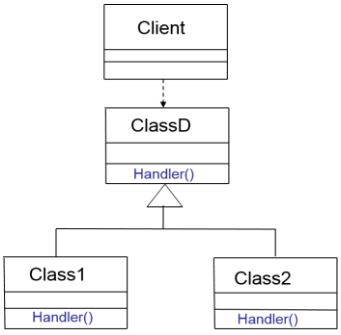

Рисунок 2. Пример неоптимального фрагмента

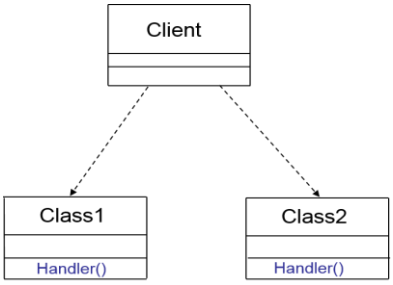

Рисунок 3. Результат использования шаблона «Цепочка обязанностей»

Цикл из отношений ассоциации, включающий класс-ассоциацию (рис. 4). Смысл этого антишаблона в следующем: если семантически связанные отношения ассоциации образуют цикл, проходящий через класс-ассоциацию, то вероятно, что одно из них избыточно. Удаление лишнего отношения ассоциации может производиться только проектировщиком.

В данном случае логично для удаления выбрать отношение ассоциации TEACING. Такой выбор определяется тем, что узнать, какие дисциплины преподает лектор, можно, осуществив навигацию через класс-ассоциацию TIMETABLE к объектам класса SUBJECT.

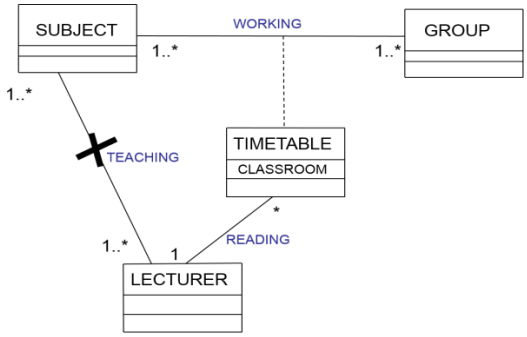

Рисунок 4. Пример антишаблона «Цикл из отношений ассоциации, включающий класс-ассоциацию»

Шаблон, позволяющий перейти от тернарной ассоциации к бинарной. Если в тернарной ассоциации имеется класс с кратностью (1) (рис. 5), то тернарную ассоциацию можно заменить на комбинацию бинарной ассоциации и класса-ассоциации (рис. 6).

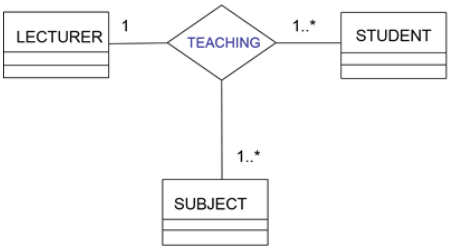

Рисунок 5. Пример тернарной ассоциации

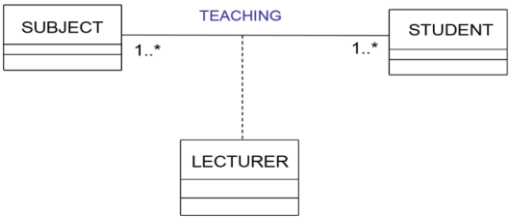

Рисунок 6. Замена тернарной ассоциации на бинарную и класс-ассоциацию

Антишаблон проектирования «клякса». Такая ситуация возникает в том случае, когда некоторый класс монополизирует в себе как сами данные, так и их обработку. То есть, этот антишаблон характеризуется наличием одного сложного класса («кляксы»), включающего множество полей и методов, в то время как ассоциированные с ним классы содержат поля и, возможно, небольшое число методов. Суть антишаблона «клякса» заключается в том, что, скорее всего, методы класса-кляксы могут и должны быть распределены по ассоциированным классам, содержащим связанную с работой данных методов информацию.

4. Инструментальное средство оптимизации ДК

В существующих CASE-средствах обычно присутствуют упрощенные средства валидации ДК, фиксирующие только связанные исключительно с правилами постро-

ения диаграмм грубые ошибки, и полностью отсутствуют средства оптимизации. Последнее вполне объяснимо, поскольку общепринятых правил оптимизации ДК не разработано. Но это не мешает использовать существующие методы описания ДК, с одной стороны, и набор правил оптимизации (естественно не полный), с другой.

Для расширения CASE-средств возможностями валидации и оптимизации ДК необходимо осуществить следующее:

1. Разработать метод поиска неоптимальных фрагментов ДК, используя имеющиеся шаблоны и анти-шаблоны проектирования.

2. Предложить в ряде случаев способы автоматической замены найденных фрагментов более оптимальными.

В качестве базового CASE-средства рассмотрен мощный пакет Rational Software Architect компании IBM, работающий на платформе Eclipse. Он поддерживает все основные элементы ДК, включая интерфейсы, классы-ассоциации, области видимости и т.д. Написан Rational Software Architect на языке Java и довольно легко расширяется с помощью плагинов.

Для поиска ошибочных и неоптимальных фрагментов необходимо знать внутреннее для CASE-средства представление ДК. Для хранения ДК UML в RSA используется формат XMI (XML Metadata Interchange) — стандарт для обмена метаданными с помощью языка XML [11]. Файл, создаваемый в формате XMI, чаще всего применяется как раз для обмена диаграммами UML между различными приложениями. В нем хранится информация о диаграммах в формате XML.

Коротко остановимся на структуре XMI-файла [12]. После корневого тега следуют теги **<packagedElement>,** соответствующие различным элементам диаграммы. Каждый элемент диаграммы обязательно имеет атрибуты **xmi:type** и **xmi:id.**

Для *класса* (**xmi:type**="uml:Class") и интерфейса (**xmi:type**="uml:Interface") обязателен атрибут **name** — имя класса. Поля класса помечены дочерним тегом **<ownedAttribute>**, атрибутами которого являются **xmi:type**="uml:Property", **xmi:id**, название **name** и область видимости **visibility.** Методы класса помечены дочерним тегом **<ownedOperation>**, атрибутами которого являются **xmi:type**=" uml:Operation", **xmi:id**, название **name**, параметры метода представляются в виде дочерних элементов <ownedParameter> с атрибутами **xmi:type**="uml:Parameter", **xmi:id** и названием **name.** Шаблон фрагмента XMI-кода, описывающего класс, приведен ниже.

```
<packagedElement
 xmi:type="uml:Class"
xmi:id="class ident"
 name="class name">
    <ownedAttribute
```

```
 xmi:type="uml:Property"
 xmi:id=" attribute ident"
name="attribute name"
 visibility="access modificator"/>
      <ownedOperation
           xmi:type="uml:Operation"
           xmi:id="operation ident"
           name="operation name">
      <ownedParameter
             xmi:type="uml:Parameter"
             xmi:id=" parameter ident"
             name="parameter name"/>
      </ownedOperation>
  </packagedElement>
```

Отношение *обобщения* представляется дочерним элементом в классе-наследнике с тегом **<generalization>**, атрибутами которого являются **xmi:type**="uml:Generalization", **xmi:id** и **general —** идентификатор xmi:id класса-предка.

Отношение *реализации* представляется отдельным элементом с тегом **<packagedElement>**, атрибутами которого являются **xmi:type**="uml:Realization", **xmi:id**, **supplier** — xmi:id интерфейса-поставщика, **client** — xmi:id класса-потребителя. Также у класса-потребителя появляется дополнительный атрибут **clientDependency**, в котором хранится идентификатор xmi:id отношения реализации.

Отношение *зависимости* представляется отдельным элементом с тегом **<packagedElement>**, атрибутами которого являются **xmi:type**="uml:Dependency", **xmi:id**, название **name**, **supplier** — xmi:id класса-поставщика, **client** – xmi:id класса-потребителя. Также у класса-потребителя появляется дополнительный атрибут **clientDependency**, в котором хранится идентификатор отношения зависимости.

Отношение *ассоциации* представляется отдельным элементом с тегом **<packagedElement>**, атрибутами которого являются **xmi:type**="uml:Association", **xmi:id**, название зависимости **name**, **memberEnd** — разделенные пробелом xmi:id дочерних элементов классов-участников ассоциации. Эти элементы представляются тегом **<ownedAttribute>** и имеют атрибуты **xmi:type**="uml:Property", **xmi:id**, название **name**, область видимости **visibility**, xmi:id другого участника ассоциации **type**, идентификатор отношения ассоциации association. В дочерних элементах представлены кратности ассоциации с тегами **<upperValue>** и **<lowerValue>.**

Отношения *агрегирования и композиции* представляются так же, как и отношение бинарной ассоциацией, но для них в дополнительных дочерних элементах классов-целого, отмеченых тегом **<ownedAttribute>,** добавляется атрибут **aggregation**, который принимает значение "shared" для агрегации и "composite" для компо-

зиции. В самом элементе ассоциации появляется дочерний элемент **<ownedEnd>**, содержащий аналогичную информацию для класса-части.

Класс-ассоциация представляется отдельным элементом с тегом **<packagedElement>** и атрибутами **xmi:type**="uml:AssociationClass", **xmi:id,** названием **name, memberEnd** — разделенные пробелом xmi:id дополнительных дочерних элементов класса ассоциации. Дочерние элементы класса ассоциации аналогичны элементам класса. Дополнительными дочерними элементами являются элементы с тегом **<ownedEnd>**, они аналогичны одноименным элементам отношений агрегирования и композиции.

5. Описание представления ДК

Для реализации алгоритмов поиска различных фрагментов диаграмм (см. табл.) избыточная форма хранения в виде стандартизированного XML-документа неэффективна. Предлагается использовать другую форму хранения, ориентированную на более быстрый поиск элементов ДК. Опишем ее.

ДК можно представить в виде набора классов (в каждом классе содержится название, атрибуты и методы) и совокупности направленных невзвешенных графов для всех типов отношений. Такое представление возможно, так как отношения между классами и интерфейсами — обобщение, ассоциация, агрегирование, композиция, реализация и зависимость — друг с другом не связаны. Таким образом, ДК можно реализовать в виде совокупности графов, вершинами которых будут классы (интерфейсы), а дугами — отношения. В каждом графе набор вершин будет одним и тем же; все графы будут ориентированными.

В качестве структуры данных для представления графов отношений в ДК следует использовать не массивы, а списки, так как в большинстве случаев графы отношений разрежены, т. е. число дуг в них невелико. Непосредственно базовый элемент и список смежных к нему целесообразно представить в виде отображения, роль ключа в котором играет идентификатор базового класса, а значением является список идентификаторов смежных классов.

Что хранится в списках смежности? Для графа отношений обобщения — это классы-предки для потомка, для графа отношения реализации — интерфейсы, реализуемые классом, для графа отношения зависимости — классы-источники для класса клиента, для графа отношения агрегирования и композиции — класс-целое для класса-части. Поскольку отношение ассоциации не ориентировано, оно будет представляться двумя отношениями; т. е. отношение ассоциации в графе дублируется. Класс- ассоциация представляется как обычный класс, связанный с классами, его порождающими. С точки зрения выявления структурных ошибок в диаграммах классов UML кратности отношений ассоциации не важны, и их можно опустить. Во

избежание многократного копирования информация о полях и методах класса выносится в отдельный список, объектами которого будут являться объекты-концепты (классы и интерфейсы) с обязательным указанием полей и методов.

Данная форма представления особенно удобна для обнаружения таких ошибок, как зацикливание, множественное наследование или агрегирование и несвязность фрагментов.

Отметим тот факт, что преобразование из формата XMI в форму многоуровневых связанных списков сложности не представляет. Соответствующий алгоритм, опирающийся на стандартные возможности парсеров, разработан.

Поскольку общее число элементов в диаграммах классов сравнительно небольшое (не более нескольких тысяч), задача поиска ошибочных и неоптимальных фрагментов при такой форме представления легко решается с помощью переборных алгоритмов.

Структурная схема, описывающая результат добавления плагина валидации и оптимизации в состав CASE-средства Rational Software Architect, представлена на рис. 7. На схеме показаны дополнительная функциональность и используемые данные.

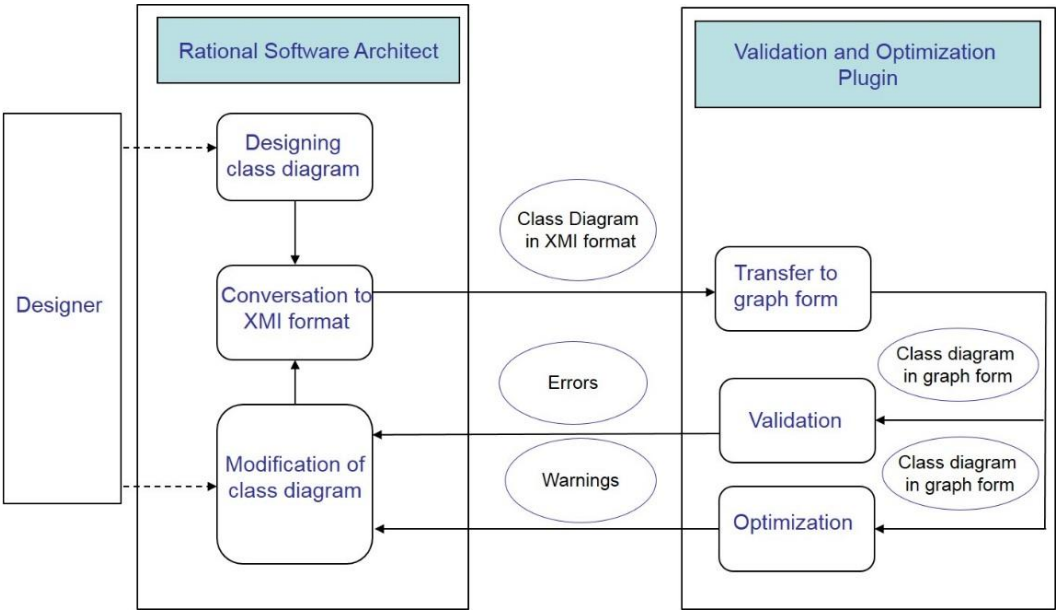

Рисунок 7. Структурная схема, описывающая работу CASE-средства

6. Заключение

В статье описан подход к оптимизации моделей программных систем на этапе проектирования. Этот подход заключается в автоматическом поиске ошибочных и неэффективных фрагментов. Для этих целей могут использоваться как шаблоны проектирования, так и антишаблоны. Информация об обнаруженных неэффективных фрагментах передается проектировщику, который и принимает решение о модификации модели. Кроме того, система может рекомендовать проектировщику применить определенные преобразования, а в ряде случаев проводить трансформации автоматически.

Актуальность подхода определяется тем фактом, что верификация и оптимизация модели проводятся уже на этапе проектирования, что позволяет в дальнейшем минимизировать время процессов отладки и рефакторинга кода.

Предлагается реализация инструментального средства оптимизации ДК для пакета Rational Software Architect компании IBM. Поскольку описание ДК в Rational Software Architect представлено в широко распространенном для передачи диаграмм UML формате XMI, разработанное средство оптимизации может быть использовано и для других CASE-средств, поддерживающих этот формат. Для реализации алгоритмов поиска фрагментов ДК предлагается использовать не XMI, а другую форму хранения — многоуровневую систему связанных списков.

Литература

[1] *Rumbaugh J., Jacobson I., Booch G.* The Unified Modeling Language. Reference Manual. — Addison-Wesley, Reading, MA, 1998.

[2] *Gamma E., Johnson R., Helm R., Vlissides J.* Design Patterns. Elements of Reusable Object-Oriented Software. — Addison-Wesley, 2001.

[3] *Berardi D., Calvanese D., Giacomo G. D.* Reasoning on UML Class Diagram // *Artificial Intelligence*. 2005. Vol. 168. P. 70–118

[4] *Sergievskiy M., Konkin A.* Optimization of UML class diagrams via description logic // *Cloud of Science*. 2017. Vol. 4. No. 3. P. 465–479.

[5] *Sergievskiy M.* Description logic application for UML class diagrams optimization // *International Journal of Advanced Computer Science and Applications*. 2017. Vol. 8. No. 1. P. 268–272

[6] *Sergievskiy M.* N-ary relations of association in class diagrams: design patterns // *International Journal of Advanced Computer Science and Applications*. 2016. Vol. 7. No. 2. P. 265–268

[7] *Sergievskiy M., Kirpichnikova K.* Optimizing UML Class Diagrams // *ITM Web of Conferences*. 2018. Vol 18. P. 03003.

[8] *Nikulchev E., Deryugina O.,* Model and criteria for the automated refactoring of the UML class diagrams // *International Journal of Advanced Computer Science and Applications*. 2016. Vol. 7. No. 12. P. 76–79

[9] *Cali A., Clavanese D., Giacomo G. D., Lcnzerini M.* A Formal framework for reasoning on UML class diagram // 13th International Symposium on Methodologies for Intelligent Systems, 2002. (IS-MIS 2002).

[10] *Brown W., Malveau R., McCormick III H., Mowbray T.* AntiPatterns. Refactoring software, architectures, and projects in crisis. — John Wiley & Sons, Inc., 1998.

[11] *Григорьев А.В., Кропотин А.А., Овсянникова Е.О.* Проблема поиска противоречий на диаграммах классов UML // Proc. of International scientific-practical conference on Modern problems and ways of their solution in science, transport, production and education'2012. http://www.sworld.com.ua/konfer29/721.pdf

[12] Materials of OMG (2005) http://www.omg.org/spec/XMI/2.1/

***Авторы*:**

Максим Владимирович Сергиевский — кандидат технических наук, доцент, доцент кафедры «Системный анализ», Национальный исследовательский ядерный университет МИФИ

Ксения Константиновна Кирпичникова — бакалавр, кафедра «Финансовый мониторинг», Национальный исследовательский ядерный университет МИФИ

Validating and Optimizing UML Class Diagrams

M. V. Sergievskiy, K. K. Kirpichnikova

National Research Nuclear University MEPhI
Kashirskoe highway, 31, Moscow, Russia, 115409

e-mail: sermax@yandex.ru

Abstract. Class diagrams play a very important role in the design process as they are used to construct the system software model. Modern CASE tools, which are basic object-oriented development tools, cannot be used to optimize UML diagrams. We will explain how, based on the use of design patterns and anti-patterns, class diagrams can be tested and optimized. Some conversions can be done automatically; others will identify potential weaknesses and provide recommendations. For this purpose, a plugin has been developed that analyzes an XML file containing a description of class diagrams. Given that the XMI view of class diagrams in Rational software Architect contains redundant information, a better view is selected to facilitate the implementation of algorithms for finding suboptimal parts of class diagrams.
Key words: UML, class diagram, design patterns, anti-patterns, CASE tool, Rational Software Architect, XMI.

References

[1] *Rumbaugh J., Jacobson I., Booch G.* (1998) The Unified Modeling Language. Reference Manual. Addison-Wesley, Reading, MA.

[2] *Gamma E., Johnson R., Helm R., Vlissides J.* (2001) Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley.

[3] *Berardi D., Calvanese D., Giacomo G. D.* (2005) *Artificial Intelligence*, 168:70–118.

[4] *Sergievskiy M., Konkin A.* (2017) *Cloud of Science*, **4**(3):465–479.

[5] *Sergievskiy M.* (2017) *Intern. J of Adv. Computer Science and Applications*, **8**(1):268–272.

[6] *Sergievskiy M.* (2016) *Intern. J of Adv. Computer Science and Applications*, **7**(2):265–268.

[7] *Sergievskiy M., Kirpichnikova K.* (2018) *ITM Web of Conferences*, 18:03003.

[8] *Nikulchev E., Deryugina O.* (2016) *International Journal of Advanced Computer Science and Applications*, **7**(12):76–79.

[9] *Cali A., Clavanese D., Giacomo G. D., Lcnzerini M.* (2002) A Fomial Framework for Reasoning on UML Class Diagram. In Proc. of the 13th Int. Sym. on Methodologies for Intelligent Systems (IS-MIS 2002).

[10] *Brown W., Malveau R., McCormick III H., Mowbray T.* (1998) AntiPatterns. Refactoring software, architectures, and projects in crisis. John Wiley & Sons, Inc.

[11] *Grigoriev A. et al.* (2012) The Problem of Detecting Consistencies on UML Class Diagrams. In SWorld-18-27 Dec. 2012 http://www.sworld.com.ua/konfer29/721.pdf. [In Rus]

[12] http://www.omg.org/spec/XMI/2.1/.