Санкт-петербургский государственный университет
Математико-механический факультет

Кафедра системного программирования

Обзор реализации механизма циклической разработки диаграмм классов и программного кода в современных UML-средствах

Курсовая работа

Бусыгина Мария
361 группа

Научный руководитель:
к.ф.-м.н., доцент кафедры системного программирования
Кознов Д.В.

Санкт-Петербург
2012

Содержание

Введение

Цель данной работы – проанализировать готовность современных UML-пакетов к промышленному применению механизма циклической разработки (round-trip engineering). Рассматривается случай циклической разработки UML-диаграмм классов – Java-кода. В работе рассмотрены следующие UML-пакеты: MagicDraw for UML (No Magic Inc), Poseidon for UML (Gentleware AG), VisualParadigm for UML (VisualParadigm Ltd), Rational Rhapsody (IBM). Эти пакеты находятся в начале списка популярных UML-средств, представленного на форуме uml-tools [4]. Исследование дополняет аналогичную работу, результаты которой представлены в [8] – то есть анализируются те UML-пакеты, которые не попали в работу [5]. На основании общей картины делаются выводы, которые не могли быть сделаны в [5] в виду недостатка рассмотренных UML-средств.

Данные UML-пакеты предоставляют не только возможность создания и редактирования UML-моделей, но также поддерживают генерацию программного кода по UML-моделям (прямая инженерия, forward engineering), обратную процедуру — генерацию UML-моделей по существующему коду (обратная инженерия, reverse engineering) и механизм циклической разработки (round trip engineering) — синхронизацию двух или более программных артефактов, таких как исходный код, модели, конфигурационные файлы и пр. [10]. Важно отметить, что это циклическая разработка не есть то же самое, что и прямая инженерия и обратная инженерия. Прямая и обратная инженерия представляют собой отдельную изолированную трансформацию одного программного артефакта в другой, при котором не учитывается предварительное состояние целевого артефакта. В большинстве случаев целевой артефакт создается заново, при этом, замещая предыдущую версию (если она существовала). В некоторых случаях, прямая и обратная инженерия оптимизируется таким образом, что трансформируются только измененные файлы (incremental transformation), в то время как неизменные в трансформации артефактов не участвуют.

В отличие от прямой и обратной инженерии, механизм циклической разработки предполагает синхронизацию и согласование артефактов, модели и кода, а не просто трансформацию одного артефакта в другой. Это означает, что изменения проносятся в целевой артефакт, минимально изменяя его, то есть, в идеале меняя только то, что требуется. Вся остальная часть целевого артефакта остается неизменной, в том числе и та, которая не имеет прообраза в исходном артефакте – источнике изменений. При уничтожение такой информации при синхронизации ее уже не восстановить автоматически, а нужно вводить в целевой артефакт заново или смириться с ее отсутствием. приводит к потере. Для диаграмм классов такой информацией является расположение элементов и другие графические свойства диаграмм, которые пользователь задал «вручную». Для объектно-ориентированного кода это «тела» методов и прочие элементы, не содержащиеся в UML.

На сегодняшний день значительное количество UML-пакетов поддерживают механизм циклической разработки. Несмотря на это, не существует общего подхода к его реализации. В каждом средстве реализованы различные механизмы идентификации кода и диаграмм. Более того, UML-пакеты предлагают ограниченное количество опций, вынуждая пользователей приспосабливаться к средству, а не наоборот.

1. Основные понятия

В данном разделе приводятся основные понятия, используемые в данном исследовании.

- **UML (Unified Modeling Language)** – это язык визуального моделирования, предназначенный для создания графических объектов объектно-ориентированных описаний программного обеспечения, создаваемых при разработке и сопровождении программного продукта.

- **UML-модель (UML-model)** – абстрактная модель системы, представленная с помощью графических элементов UML.

- **Диаграмма классов (class diagram**) – статическая структурная диаграмма, описывающая структуру системы. Она демонстрирует классы системы, их атрибуты, методы и зависимости между классами. В данной работе были рассмотрены диаграммы классов с точки зрения реализации и содержали классы, непосредственно используемые в программном коде.

- **Подходы к разработке, управляемые моделями (Model-Driven Software Development approaches)** – это модельно-ориентированные подходы к разработке программного обеспечения, когда модели становятся основными артефактами разработки, из которых генерируется код и другие артефакты.

- **Механизм циклической разработки (round-trip engineering)** – это синхронизация двух или более программных артефактов, таких как исходный код, модели, конфигурационные файлы (например, схемы XML-файлов) и пр. [10]

  Основные этапы механизма циклической разработки:
  1) выявление того, что артефакт был изменен;
  2) выявление того, что артефакты (например, код и модель) не согласованы;
  3) выполнение синхронизации артефактов при выявлении несогласованности;

  В результате, артефакты должны быть согласованы.

  - **Сильный механизм циклической разработки (Strict synchronization)** – механизм циклической разработки, предполагающий автоматическую синхронизацию артефактов при изменениях.

  - **Слабый механизм циклической разработки (Loose synchronization)** – механизм циклической разработки, предполагающий синхронизацию артефактов по требованию (например, по желанию пользователя или по таймеру).

2. Критерии оценки

Согласно работе [5], для оценивания UML-пакетов были использованы следующие критерии:

- используемый механизм идентификации;
- поддержка простых изменений;
- поддержка сложных изменений;
- возможности настроек;
- удобство использования.

1) **Механизм идентификации** (mapping – способ соответствия элементов синхронизируемых моделей). **От него непосредственно зависит качество реализации механизма циклической разработки** – сколь тонкие изменения он может распознавать и распространять. Следуя [9], обозначим следующие способы идентификации:

   - **По имени.** Соответствие между элементами моделей устанавливается по именам. Недостатком данного механизма является то, что при переименовании элемента в одной модели связь между соответствующими элементами в другой теряется.

   - **По уникальному идентификатору (по GUID).** Элементы, как модели, так и кода, имеют некоторый идентификатор, в случае, если идентификаторы элемента модели и элемента кода совпадают, между ними устанавливается соответствие. Благодаря тому, что идентификатор не зависит от имени объекта, удается преодолеть недостаток предыдущего подхода. Кроме того, так как идентификатор обычно спрятан и не может быть изменен пользователем, удается достичь большей надежности идентификации. Таким образом, при изменении элементов модели можно однозначно определить, какие элементы связанной модели должны быть синхронизованы. Идентификаторы можно хранить, например, в комментариях специального формата. Недостаток данного подхода возникает в некоторых средах и заключается в том, что не всегда удается установить однозначное соответствие между элементами синхронизируемых моделей, например, если при копировании элемента модели копируется и идентификатор. Если же при копировании генерируется новый идентификатор, теряется связь между объектом и его копией. Кроме того, идентификаторы нужно хранить, а также обновлять.

   - **Вынесенная идентификация.** В этом случае идентификаторы двух сущностей из модели и кода хранятся отдельно от кода и модели (в принципе, это могут быть не пары, а две группы — как с одной, так и с другой стороны может быть задана группа сущностей.) Отметим, что речь идет о конкретных ”живых” элементах модели и кода (экземпляры классов, переменные), а не об отношении на уровне метамодели и языка программирования (только типы данных). Для данного способа идентификации зачастую используются индексные файлы.

   - **По имени с различными эвристиками** (например, сверяясь по отношениям с другими объектами, просматривая состав полей объекта).

код		диаграмма	
изменение не отображается	**изменение отображается**	**изменение не отображается**	**изменение отображается**
0	0.5	0	0.5

2) **Поддержка простых изменений.**

Рассматривались добавление и удаление следующих элементов, как в коде, так и в модели: классов, методов, атрибутов, параметров операций, другие простые модификации.

код		диаграмма	
изменение не отображается	**изменение отображается**	**изменение не отображается**	**изменение отображается**
0	0.5	0	0.5

При этом модификация элементов диаграммы не означает смены имени и владельца (области видимости), а только изменение атрибутов сущности, например, параметров операций. Для реализаций механизма циклической разработки важно, чтобы все эти простейшие операции работали надежно. В противном случае возможность пользоваться данным механизмом в промышленных проектах значительно снижается.

Максимальная оценка: 10 баллов

3) **Поддержка сложных изменений**

код		диаграмма	
изменение не отображается	**изменение отображается**	**изменение не отображается**	**изменение отображается**
0	0.5	0	0.5

Алгоритм исследования также определял, были ли выполнены более сложные операции. Чем более семантически сложные операции распознает механизм синхронизации, тем более “тонко” он способен работать, в большей степени распространяя изменения и в меньшей степени перегенерируя заново фрагменты целевой спецификации. При перегенерации теряется та информация, которая в эти фрагменты была добавлена пользователем, но не содержится (и, соответственно, не синхронизирована) в исходной спецификацией. Согласно работе [5] были выделены следующие подобные операции.

- **Переименование элементов.** Важно, что при переименовании элемента в одной модели не потеряется связь с соответствующим ему элементом другой модели, иначе некоторая важная информация (например, реализация метода) может быть утрачена.

  Рассматривались переименования следующих элементов, как в коде, так и в модели: пакетов, классов, методов, атрибутов, ассоциаций.

  Максимальная оценка: 5 баллов

- **Копирование элементов.** Хотелось бы, чтобы при копировании элемента модели копировался не только элемент в данной модели, но и соответствующий ему элемент другой модели. Например, при копировании метода или класса в UML-диаграмме желательно, чтобы соответствующая

реализация в коде также была скопирована. Естественно, копии в диаграмме и в коде должны быть синхронизованы между собой. Кроме того, после завершения процесса копирования, элемент и его копия должны существовать самостоятельно и не зависеть друг от друга.

Рассматривалось копирование следующих элементов: пакетов, классов, методов, атрибутов, параметров операций, ассоциаций.

Максимальная оценка: 6 баллов

- **Перенос элементов (например, перемещение метода в другой класс).** В данном случае важно, чтобы при перемещении элемента не потерялась связь с соответствующим ему элементом другой модели. Зачастую перемещенный элемент воспринимается не как уже существующий элемент, а как новый элемент, что приводит к потере связанной с ним информации.

  Рассматривались переносы следующих элементов: классов, методов, атрибутов, параметров операций, ассоциаций.

  Максимальная оценка: 2.5 балла

- **Изменение связей между элементами: добавление, удаление ассоциаций, изменение типа ассоциаций.** Важно не только отражение внесенных изменений, но и сохранение каждой модели целостной и согласованной. Возможность получить некомпилируемый код после синхронизации делает механизм циклической разработки намного менее надежным.

  Рассматривались следующие изменения: добавление ассоциаций, удаление ассоциаций, изменение типа ассоциаций, изменение области видимости ассоциаций, а также изменения полюсов ассоциаций (множественность, роли, направленность).

  Максимальная оценка: 7 баллов

4) **Удобство использования** (**usability**). Здесь рассматривались такие аспекты как: интуитивная понятность функциональности и удобство пользовательского инструмента, качество пользовательской документации, возможности интеграции в IDE и перехода на другую среду разработки.

   Usability – степень производительности, удовлетворенности и результативности, с которыми продукт может применяться пользователями в конкретных условиях для достижения заданных целей [ISO 9241-11].

Таблица 1
Характеристики ISO/IEC 9126-1

№	характеристика	подробнее
1	Understandability	простота понимания заложенной в продукт логики
2	Learnability	простота обучения использованию продукта
3	Operability	простота оперирования
4	Attractiveness	привлекательность приложения
5	Usability compliance	соответствие официальным стандартам

По 1 баллу за каждый пункт.

Возможности настроек (“гибкость” опций, предоставленных пользователю). Все настройки, связанные с генерацией кода и отображением модели. Например:

- Опция “Merge code and model”
- Префиксы атрибутов и методов
- Выбор представления ассоциаций в коде
- Выбор представления атрибутов при синхронизации после изменений кода
- Задание параметров для форматирования кода (скобки и отступы, комментарии и пр.)
- Автоматический “красивый” layout диаграмм
- Возможность “спрятать” элементы диаграммы во избежание нагромождения

Максимальная оценка: 5 баллов

3. Рассмотренные UML-пакеты

Критерии выбора UML-пакетов для обзора основаны на данных из форума UML-forum [4].

Таблица 2
Рейтинг UML-пакетов

Место в рейтинге	UML-пакет	Компания	Поддержка round-trip
1	**Sparx Enterprise Architect**	Sparx Systems	+
2	**MagicDraw UML**	MagicDraw Inc	+
3	**UModel**	Altova	+
4	StarUml (open source)	StarUML	-
5	**VisualParadigm for UML**	VisualParadigm Ltd	+
6	Rational Rhapsody	IBM	-
7	**RationalSoftware Architect**	IBM	+
8	Modelio free editon (open source)	Modeliosoft	-
9	Papyrus UML (open source)	PAPYRUS	-
10	TOPCASED (opensource)	TOPCASED	-

В таблице 2 представлены пакеты, рассмотренные в данной работе. Помимо пакетов, MagicDraw for UML, VisualParaigm for UML и Rational Rhapsody был рассмотрен пакет Poseidon for UML, также являющийся довольно известным средством моделирования [3].

Таблица 3
Выбранные UML-пакеты

№	Название	Компания	Версия	Комментарии
1	Poseidon for UML Professional Edition	Gentleware (Germany)	6.0.2-0	коммерческий ”последователь” open source проекта ArgoUML
2	VisualParadigm for	VisualParadigm Ltd (Hong-Kong)	8.3	коммерческий

	UML Enterprise edition			
3	MagicDraw for UML	MagicDraw Inc	17.0.1	коммерческий

Полученные результаты

Во всех рассмотренных UML-пакетах используется слабый механизм синхронизации. Результаты сведены в единую таблицу.

Poseidon for UML

1) **Механизм идентификации**
   В UML-пакете Poseidon for UML используется идентификация по GUID. Идентификаторы GUID хранятся в комментариях специального формата:

```
/**
 *
 *
 * @poseidon-object-id [Im118bb72cm13760b79e1fmm7e5f]
 */
public class ClassA {
        …
}
```

   Неудобство данного способа синхронизации вызвано тем, что после добавления изменений кода в модель, то код и модель остаются не синхронизированными из-за отсутствия в коде идентификаторов, проставленных Poseidon'ом, поэтому для синхронизации кода и модели нужно дважды вызывать синхронизацию *(+22.5-2)*.

2) **Простые изменения**
   Простые изменения работают без нареканий *(+10)*.

3) **Сложные изменения**
   При переименовании элементов модели (кода) соответствующие им элементы кода (модели) создаются заново. То же самое происходит и при копировании любых элементов на UML-диаграмме *(+12.5)*.

4) **Удобство использования**
   Из рассмотренных UML-пакетов Poseidon является самым простым в плане изучения, понятности интерфейса и оперирования продуктом (+4). Несмотря на понятность, интерфейс довольно небрежен. Также нельзя не отметить сравнительно небольшое количество настроек (+1).

VisualParadigm

1) **Механизм идентификации**

В VisualParadigm for UML используется вынесенная идентификация при помощи индексных файлов. *(+22.5)*

2) **Простые изменения**

Простые изменения работают без нареканий. *(+10)*

3) **Сложные изменения**

Не работают те же модификации, что и в предыдущем UML-пакете. При переименовании элементов модели (кода) соответствующие им элементы кода (модели) создаются заново. То же самое происходит и при копировании любых элементов на UML-диаграмме *(+12.5).*

Удобство использования:

Данный UML-пакет отличается большим количеством и гибкостью настроек (+5), поэтому изучение, понятность и оперирование продуктом несколько затруднено (+2).

MagicDraw

1) **Механизм идентификации**

В MagicDraw for UML, так же как и в VisualParadigm, используется вынесенная идентификация при помощи индексных файлов. Но в данном средстве всю идентификацию необходимо задавать “вручную” для пакетов и классов, что неудобно *(+18-3).*

2) **Простые изменения**

Простые изменения работают без нареканий за исключением добавить новый параметр – из кода или из модели – то генерируется новый метод. *(+9,5)*

3) **Сложные изменения**

В данном UML-пакете некорректно отображаются некоторые модификации с ассоциациями: не отображается множественность и нельзя изменять тип ассоциаций. Также в коде не отображается перенос элементов на UML-диаграмме.

При переименовании элементов модели (кода) соответствующие им элементы кода (модели) создаются заново. Переименованные элементы нужно заново добавлять в процесс round-trip “вручную”, и соответственно, они заново создаются в коде. Переименование более мелких тоже элементов не работает. Точно такая же ситуация с копированием элементов на UML-диаграмме *(+8).*

Удобство использования

Интерфейс данного UML-пакета не отличается ни большим количеством настроек, ни легкостью в обучении, ни привлекательностью для пользователя (+2.5+2).

.

Таблица 3
Сводная таблица результатов

	MagicDraw		Visual Paradigm		Poseidon for UML	
ПОДДЕРЖКА ПРОСТЫХ ИЗМЕНЕНИЙ						
	код	**диаграмма**	**код**	**диаграмма**	**код**	**диаграмма**
Добавление класса	+	+	+	+	+	+
Удаление класса	+	+	+	+	+	+
Добавление атрибута	+	+	+	+	+	+
Удаление атрибута	+	+	+	+	+	+
Добавление метода	+	+	+	+	+	+
Удаление метода	+	+	+	+	+	+
Добавление параметров операций	+	-	+	+	+	+
Удаление параметров операций	+	-	+	+	+	+
Простые модификации	+	+	+	+	+	+
ПОДДЕРЖКА СЛОЖНЫХ ИЗМЕНЕНИЙ						
I. Переименование элементов						
пакета	-	-	-	-	-	-
класса	-	-	-	-	-	-
атрибута	+	+	+	+	+	+
метода	+	+	+	+	+	+
ассоциаций	+	+	+	+	+	+
II. Копирование элементов на UML-диаграмме						
пакет	-		-		-	
класс	-		-		-	
метод, атрибут, параметр, ассоциация	-		-		-	
III. Перенос элементов на UML-диаграмме						

класс	-		+		+	
метод, атрибут, параметр, ассоциация	-		+		+	
IV. Изменение связей между элементами						
Добавление ассоциаций	+	+	+	+	+	+
Удаление ассоциаций	+	+	+	+	+	+
Изменение типа ассоциаций	-	-	+	+	+	+
Изменения области видимости ассоциаций	+	+	+	+	+	+
Изменение полюсов ассоциаций						
множественность	-	-	+	+	+	+
роли	+	+	+	+	+	+
направленность	+	+	+	+	+	+

5. Как проходил эксперимент

1) Разработка контрольного Java-приложения

   Для анализа реализации механизма циклической разработки было написано Java-приложение, содержащее те элементы кода, для которых существуют UML-сущности. Согласно [1], рассматривались следующие сущности UML-модели:

   - Ассоциации (Association)
   - Класс ассоциаций (AssociationClass)
   - Класс (Class)
   - Тип данных (DataType)
   - Зависимость (Dependency)
   - Обобщение (Generalization)
   - Множество обобщения (GeneralizationSet)
   - Интерфейс (Interface)
   - Реализация интерфейс (InterfaceRealization)
   - Операция (Operation)
   - Пакет (Package)
   - Включение пакета (PackageImport)
   - Слияние пакетов (PackageMerge)
   - Параметр (Parameter)
   - Тип видимости (VisibilyKind)

- Пакет (Package)

Для того, чтобы проверить, правильно ли отображаются изменения из кода в UML-модель (и обратно), в код подвергался различным изменениям, связанным с каждым из данных элементов.

2) Инсталляция пакетов

В данном исследовании были рассмотрены коммерческие UML-пакеты. Вследствие этого, возникали некоторые сложности, связанные со скачиванием, установкой и настройкой пакетов. Эти проблемы вызваны тем, что пакеты доступны только в trial версии. (Для обхода ограничений можно внести изменения в реестр Windows, чтобы была возможность чистую переустановку программного пакета.)

3) Изучение UML-пакетов

Данный пункт включает в себя следующие этапы:

- изучение главных для исследования возможностей пакетов (forward and reverse engineering, round-trip engineering);
- ознакомление с пользовательской документации (в тех случаях, когда интерфейс пакета не оказывался интуитивно понятным).

4) Собственно эксперимент

Этапы:

- выбор UML-пакетов (см.п.3)
- инсталляция UML-пакетов
- написание контрольного Java-приложения
- изучение UML-пакетов
- исследование работы механизма циклической разработки в выбранных UML-пакетах
- получение и анализ результатов

5) Анализ результатов

Для наглядности и полноты представления результаты данного исследования были сведены в единую итоговую таблицу вместе с результатами, полученными в исследовании [5].

Таблица 4
Результат оценки UML-пакетов

	MagicDraw	**Enterprise Architect**	**Rational Software Archtect**	**VisualParadigm for UML**	**Poseidon for UML**	**UModel**
Идентификация	**14**	**20**	**21,5**	**22,5**	**20.5**	**26**
Простые изменения	**9**	**8**	**9**	**10**	**10**	**10**
Сложные изменения	**8**	**12**	**12,5**	**12,5**	**12.5**	**16**

Удобство использования	**2.5**	**5**	**4**	**2**	**4**	**5**
Настройки	**2**	**2**	**5**	**5**	**1**	**2**
Итого	**35.5**	**47**	**52**	**52**	**48**	**59**

6. Выводы

- Лучшее средство, поддерживающее циклическую разработку – UModel.
- К сожалению, данная функциональность либо слишком сложна и неочевидна.
- В некоторых UML-пакетах этот механизм не поддерживается вовсе.
- Механизм циклической разработки может быть ограниченно использован в серьезных промышленных проектах.

6. Литература

[1] UML 2.0 Infrastructure Specification, September, 2004, http://www.omg.org/

[2] OMG Unified Modeling Language (UML) Vendor Directory, http://uml-directory.omg.org/vendor/list.htm

[3] UML Products by Company, http://www.objectsbydesign.com/tools/umltools_byCompany.html

[4] UML-forum, http://www.uml-forum.com/tools.htm

[5] Холтыгина Н.А., Кознов Д.В. Обзор реализации механизма циклической разработки диаграмм классов и программного кода в современных UML-средствах. Системное программирование. Вып. 5: Сб. статей / Под ред. А.Н.Терехова, Д.Ю.Булычева. СПб.: Изд-во СПбГУ, 2010 С. 76-94.

[6] Boklund A., Mankefors-Christiernin S., Johansson C., Lindell H. AmComarative Study of Forward and Reverse Engineering in UML Tools In. Proceedings APPLIED COMPUTING 2007, Salamanca, Spain, 18–20 February 2007.

[7] Sendall S., Kaster J. Taming Model Round-Trip Engineering // Proceedings of Workshop on Best Practices for Model-Driven Software Development, 2004.

[8] B. Scholtz An Investigation into the Learnability of Object-Oriented CASE Tools for Computing Education, 2007, Nelson Mandela Metropolitan University