УДК 004.4.242

Ерохин М.В.

Московский авиационный институт (национальный исследовательский университет), г. Москва, Россия

КОНВЕРСИЯ ИЗ UML В ЯЗЫК ПРОГРАММИРОВАНИЯ

АННОТАЦИЯ

Проводится сравнение инструментов генерации исходного программного кода на основе UML-диаграмм. Предлагается единый подход к преобразованию структурных и поведенческих UML-диаграмм в тексты на языках программирования через представление по спецификации XMI. Используются метамодели языка UML.

КЛЮЧЕВЫЕ СЛОВА

UML; диаграммы; программирование; разработка программного обеспечения; генерация кода; XMI.

Erokhin M.V.

Moscow Aviation Institute (National Research University), Moscow, Russia

CONVERSION FROM THE UML TO A PROGRAMMING LANGUAGE

ABSTRACT

Tools for generating of program source code from UML-diagrams are compared. A unified approach to the conversion of structured and behavioral UML-diagrams into programming language source code is considered, being performed through the XMI presentation. Meta-models of the UML are used.

KEYWORDS

UML; diagrams; programming; software development; code generation; XMI.

С момента появления компьютеров и ЭВМ люди всегда старались оптимизировать процесс создания Информационных Систем (ИС). По сложившейся практике непременным этапом этого процесса выступает проектирование, в рамках которого стремятся построить полные и непротиворечивые функциональные и информационные модели создаваемой ИС и её программного обеспечения. Накопленный к настоящему времени опыт проектирования ИС показывает, что это логически сложная, трудоемкая и длительная по времени работа, требующая высокой квалификации участвующих в ней специалистов. По настоящее время проектирование ИС выполняется в основном на интуитивном уровне с применением неформализованных методов, основанных на искусстве, практическом опыте, экспертных оценках и дорогостоящих экспериментальных проверках качества функционирования ИС. К тому же в процессе создания и функционирования ИС информационные потребности пользователей могут изменяться или уточняться, что еще более усложняет разработку и сопровождение таких систем. Процесс разработки таких ИС основывается на моделировании деятельности предприятия, описании организации и методов ведения их бизнеса, построении архитектуры системы и структуры баз данных, обосновании системы математических моделей и алгоритмов, реализации пользовательского интерфейса и выборе технических средств. Для такого моделирования широко применяется язык UML, предложенный консорциумом OMG (Object Management Group) [1].

UML является языком широкого профиля, это — открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования.

В настоящее время крупные программы сначала проектируются и только после этого записываются на языке программирования. При использовании UML проектирование сопровождается разработкой модели, различные “виды” (аспекты) которой представляются в виде UML-диаграмм. Диаграммы UML сохраняют детали проектных решений, обеспечивают сотрудничество при коллективном проектировании, способствуют составлению эксплуатационной

документации и облегчают доработку программы в ходе её сопровождения. UML позволяет выразить не только структуру программы и обрабатываемых ею данных, но и её поведение, вплоть до детальных алгоритмов. Однако доступные инструменты проектирования на UML, такие как StarUML [2] или Rational Software Architect [3], не предоставляют возможности автоматически преобразовывать любые алгоритмы, выраженные в диаграммах UML, ни в исполняемые модули на машинном языке, ни в тексты на языках программирования. Разработчик программы вынужден разбираться в моделях, которые написал проектировщик, и только потом «вручную» переписывать текст на языке UML в текст на языке программирования, а затем компилировать программу. На этом этапе происходит двойная работа и теряется время.

Однако, процесс разработки может быть упрощен посредством генерации программного кода из UML-диаграмм.

Генерация кода предусматривалась еще с первых версий Rational Rose (из диаграммы классов генерировались пустые классы, которые впоследствии нужно было определять), однако минус таких решений, что в каждом редакторе нужно по-своему разбираться в генерации кода и в построении UML-моделей. И сам результат кодогенерации сильно разнится от одного редактора к другому.

Например, StarUML 2 генерирует код по таким правилам:

1. UMLPackage - преобразуется в директорию в проекте C++.
2. UMLClass - преобразуется в обычный класс C++. Объявление этого класса в .h файле. Определение в .cpp. Видимость класса по умолчанию protected. Поддерживаются модификаторы isFinalSpecialization и isLeaf, которые преобразуются в ключевое слово final, введенный в C++, начиная со стандарта C++11. Автоматически генерирует пустой конструктор по умолчанию. Также поддерживаются шаблоны.
3. UMLAttribute - преобразуется в поле класса. Видимость поля по умолчанию protected. Имя атрибута преобразуется в имя поля. Тип атрибута преобразуется в тип поля. Если атрибут имеет модификатор multiplicity, то в C++ поле класса будет вектором. isStatic преобразуется в статическое поле класса. isLeaf преобразуется в ключевое слово final. Значением атрибута defaultValue будет инициализировано поле данных.
4. UMLOperation - преобразуется в метод. Видимость метода по умолчанию protected. Поддерживается модификатор isAbstract, в таком случае метод будет виртуальным. Поддерживается модификатор isStatic. В случае если нет возвращаемого значения, устанавливается void. Поддерживается модификатор isReadOnly, устанавливается модификатор const в C++ коде.
5. 5.UMLInterface - преобразуется в C++ класс. Объявление класса в .h файле. Видимость класса по умолчанию protected. Все методы класса будут чисто виртуальными.
6. 6.UMLEnumeration - преобразуется в C++-перечисление. Определяется в отдельном .h файле. Видимость по умолчанию protected. Литералы в перечислении преобразуются в C++-литералы.
7. UMLAssociationEnd - преобразуется в C++ поле. Видимость поля по умолчанию protected. Если ассоциация одна из таких: 0..*, 1..*, *, тогда будет использован C++ vector (std::vector<T>). Значение по умолчанию также поддерживается.
8. UMLGeneralization & UMLInterfaceRealization - преобразуется в наследование C++. Поддерживается как “UMLClass to UMLClass” и “UMLClass to UMLInterface”.

Visual Studio корпорации Microsoft, например, поддерживает генерацию текста программы по тексту UML только для языка C# и использует ограниченный набор UML-типов (класс, интерфейс, перечисление). Все остальное пользователь может сам настроить с помощью шаблонов. Шаблон – это файл в формате XML который определяет, как генерировать ту или иную UML-сущность. Недостаток шаблонов в том, что их никак нельзя использовать в других UML редакторах.

StarUML поддерживает много языков и генерирует пустые классы и функции. Определение, файл проекта и точку входа в приложение следует создать самому пользователю. Недостаток всех существующих инструментов для кодогенерации в том, что логику исполнения, поведение никто из них не генерирует. В докладе предлагается решение, которое сможет учесть эти проблемы и решить их: а именно: универсальный способ преобразования диаграмм UML в тексты на языках программирования через представление на языке XMI. Основные инструменты проектирования на UML обеспечивают преобразование диаграмм в это представление. Процесс разработки должен выглядеть так: проектировщик создает UML-диаграммы системы в любой программе проектирования, поддерживающей экспорт UML-моделей в XMI [4]; предлагаемая программа принимает на вход этот XMI-файл и генерирует C++ код на основе диаграмм.

Например, диаграмму активности:

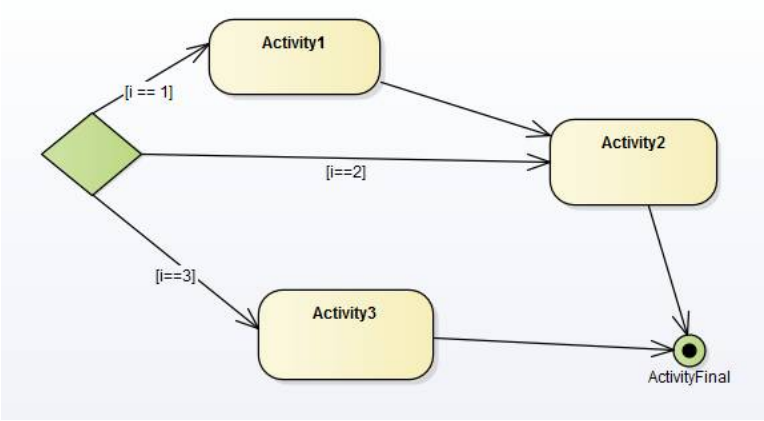

Рис.1. Диаграмма активности

Можно интерпретировать так:

```
if ( i == 0 ){
        // Action_1
        // Action_2
} else if ( i == 2 ){
        // Action_2
} else if ( i == 3 ){
        // Action_3
}
```

Спецификация XMI была разработана консорциумом OMG на основе языка XML как средство стандартного представления UML-моделей, не зависящее от графического оформления диаграмм. XMI часто применяется как формат обмена UML-моделями между программами, обеспечивающими работу с UML, но не совместимым между собой по внутреннему представлению моделей, так и по формату их хранения в виде файлов. Формат XMI поддерживают, кроме StarUML и Rational Software Architect, также другие программы, в частности ArgoUML [5] и Enterprise Architect [6].

XMI является расширением XML как языка. В соответствии со стандартом XML первая строка файла имеет такой вид: <?xml version="1.0" encoding="windows-1252"?>. Расширение XML — это конкретная грамматика, созданная на базе XML и представленная словарем тегов и их атрибутов, а также набором правил, определяющих какие атрибуты и элементы могут входить в состав других элементов.

В актуальной версии UML существует более 12 диаграмм, однако генерацию исходного кода следует, в первую очередь, обеспечить для диаграмм классов и последовательности. Дополнительные диаграммы можно будет использовать для оптимизации кода.

Разрабатываемый конвертер позволит расширить применение UML, ускорить разработку программ и повысить их качество.

Программа принимает в качестве входных данных XML-файл в формате XMI. На выходе будут файлы исходного кода на языке C++. Сам транслятор работает в два этапа:

1. Обработка входного файла;
2. Генерация исходного кода.

Так как программа кодогенерации работает с XMI файлами, то и ее функциональность ограничивается способностью формата XMI выразить алгоритм, представленный средствами UML. Рассмотрим формат XMI ближе. Файл начинается с открывающего тэга <xmi:XMI>, все что будет в этом файле, будет находиться внутри этого тэга. У этого тега есть 3 атрибута: xmi:version, которая указывает используемую версию стандарта XMI и 2 атрибута: xmlns:uml и xmlns:xmi указывающие ссылки на документацию UML и XMI соответственно. С помощью этих атрибутов человек читающий этот файл или программа могут знать какой именно формат XMI и UML используется в файле.

Далее как потомок узла <xmi:XMI> присутствует тэг <uml:Model>, именно он в первую очередь интересен для кодогенерации, так как содержит всю информацию по модели. Все диаграммы, которые пользователь создаст у себя в редакторе UML окажутся потомками элемента <uml:Model>, например диаграмма классов будет открываться тэгом <packagedElement xmi:type="uml:Class">, у диаграммы активности в атрибуте xmi:type будет «uml:Activity», у диаграммы взаимодействий «uml: Collaboration». И так далее XMI файл описывает всю UML-модель, используя метамодель UML.

UML-метамодель — это модель, которая описывает язык UML — в частности, она описывает классы, атрибуты, ассоциации, пакеты (packages), сотрудничество (collaborations), варианты

использования (use cases), актеры, сообщения, состояния и другие понятия языка UML. Метамодель сама может быть написана в UML.

Приставка "мета" означает, что метамодель описывает модель модели. Кроме того, XML используется в данном случае как метаязык (язык, на котором описывается другой язык). Метамодель UML опубликована в спецификации UML. Более конкретно, XMI использует "UML Model Interchange", описанную в главе 5 спецификации UML.

Одним из центральных понятий в UML, по крайней мере — в диаграммах классов — является само понятие "class". В метамодели это понятие моделируется метаклассом Class, наследующим от абстрактного метакласса Classifier. Classifier является родительским для класса, интерфейса и типа данных. Цепочку наследования продолжают: GeneralizableElement, который представляет все концепции, которые могут быть генерализованы (то есть — унаследованы от других); ModelElement, который представляет все абстракции в модели (такие, как пространство имен, ограничения, класс), и, наконец, Element, самый верхний метакласс. Каждый из этих метаклассов имеет атрибуты, из которых Class наследует.

Обработка входного файла включает в себя проход по XML-дереву файла с записыванием в память всех классов, интерфейсов, отношений, типов данных. Для того, чтобы максимально удобно описать UML-модель, используется четырехслойная архитектура метамоделирования. Она позволит структурировать и сохранить определённый уровень гибкости и расширяемости.

Ядром является мета-мета модель M3 на верхнем уровне. Она определяет язык, используемый для создания метамоделей, называемых M2-моделями. Наиболее ярким примером модели 2-го уровня MOF является метамодель UML: модель, которая описывает сам язык UML (именно этот слой соответствует классам в моем конвертере). Эти M2-модели описывают элементы M1 слоя: M1-модели. Это могут быть, например, модели, написанные на UML. Последний слой — M0-слой или слой данных. Он используется для описания объектов реального мира.

Использование этой модели позволяет достигнуть нужного уровня расширяемости и универсальности. Реализация метамодели сейчас обеспечивает более устойчивую и гибкую программу, которую потом не придется переделывать ради добавления небольшой возможности. Так как XMI использует эту архитектуру, большинство средств UML также могут быть представлены в формате XMI. Это касается и структурных диаграмм, и диаграмм поведения.

Для того, чтобы отличать различные объекты, язык XMI вводит атрибут xmi:id – уникальный буквенно-числовой идентификатор. Он есть у всех объектов метамодели UML: у класса, метода, поля, ограничения, диаграммы, комментария, сообщения, отношения. Таким образом, XMI может однозначно указать на какой объект ссылается какая-нибудь функция или какие сообщения принадлежат к combined fragment'у в диаграмме взаимодействий.

Вот так в формате XMI представляется combined fragment типа «цикл»:

```
<fragment xmi:type="uml:CombinedFragment" xmi:id="EAID_EDF362B176E1" name="loop1" visibility="public" interactionOperator="loop">
<covered xmi:idref="EAID_16F19A5BA514"/>
 <covered xmi:idref="EAID_ D4C9A4D27CC3"/>
  <operand xmi:type="uml:InteractionOperand" xmi:id="EAID_ EDF362B176E1">
   <guard xmi:type="uml:InteractionConstraint" xmi:id="EAID_EDF362B176E1">
    <specification xmi:type="uml:OpaqueExpression" xmi:id="EAID_EDF362B176E1" body="Object2 != Null"/>
  </guard>
 <fragment xmi:type="uml:OccurrenceSpecification" xmi:id="EAID_ D4C9A4D27CC3" covered= "EAID_D4C9A4D27CC3"/>
…
</operand>
</fragment>
```

Видно, что для каждого элемента указывается уникальный идентификатор. У фрагмента указывается зона видимости и тип взаимодействия, а также имя, которое ему присвоил пользователь. Таким образом представляется любая диаграмма UML.

Так как UML предоставляет более широкие возможности по моделированию и описанию объектов, классов и отношений, на этапе генерации кода необходимо решить ряд проблем по интерпретации UML-сущностей в программу на языке C++.

Например, UML предоставляет 7 видов отношений между классами:

1. Ассоциация – отношение между классами объектов, которое позволяет одному экземпляру объекта вызвать другого, чтобы выполнить действие от его имени. Фактически, если объект А ассоциирован с объектом Б, означает, что объекту А известен объект Б. В C++ это отношение можно интерпретировать через указатель на объект Б в объекте А;
2. Зависимость – отношение между классами объектов, которое говорит, что один объект использует другой (зависит от него). В C++ это отношение явно никак не объявляется. Это

отношение имеет место в том случае, если независимый класс является параметром какого-нибудь метода зависимого класса или независимый класс используется как локальная переменная внутри методов зависимого класса;

3. Агрегация – отношение «часть-целое» между двумя равноправными объектами, когда один объект (контейнер) имеет ссылку на другой объект. Оба объекта могут существовать независимо: если контейнер будет уничтожен, то его содержимое — нет. В C++ это интерпретируется также через указатель на объект;
4. Композиция (агрегирование по значению) – более строгий вариант агрегирования, когда включаемый объект может существовать только как часть контейнера. Если контейнер будет уничтожен, то и включённый объект тоже будет уничтожен. В C++ включаемый объект будет просто полем класса-контейнера;
5. Наследование – отношение «является». Данное отношение означает, что если объект Эллипс на следован от объекта Фигура, то объект Эллипс является объектом Фигура. Данное отношение полностью поддерживается языком C++;
6. Шаблонный класс – также отношение между классами, поддерживаемое языком C++. Означает, что шаблонный класс является классом, с помощью которого можно сгенерировать обычный класс, подставив свой тип данных;
7. Определение интерфейса – так как UML разделяет классы и интерфейсы, то наследование и реализацию интерфейса как отношения он также разделяет. В C++ семантически наследование и реализация интерфейса не отличаются.

Также UML определяет несколько видов многочисленности отношения с каждой стороны:

«0..1» - нет экземпляров или один. Реализовать это в C++ можно с помощью boost::optional или std::optional начиная со стандарта C++17. Либо хранить указатель, но следить за его использованием.

«1» - один экземпляр. В C++ реализация через ссылку.

«0..*» - ноль или больше экземпляров. В C++ можно использовать std::vector.

«1..*» - один или больше экземпляров. В C++ можно использовать std::vector с проверкой в конструкторе на непустой массив.

Учитывая все вышесказанное, можно заключить, что на языке UML будущее программное обеспечение можно выразить довольно глубоко и разносторонне. Можно выражать не только структурные, но и диаграммы поведения, а это открывает широкие возможности для проектировщиков и программистов, в частности, для обмена проектами и быстрой разработки сложных программ в области обработки больших данных. Также можно утверждать, что выбранный формат представления и обмена UML-моделями XML Metadata Interchange в полной мере обеспечивает выражение всей UML-модели в текстовой форме. XMI также поддерживает все типы диаграмм, ограничения и большинство вещей, которые поддерживает сам язык моделирования UML.

Литература

1. OMG Unified Modelling Language (OMG UML). Version 2.5. URL: http://www.omg.org/spec/UML/2.5/PDF.
2. StarUML is a UML tool by MKLab. URL: http://staruml.io/.
3. IBM® Rational® Software Architect Designer (RSAD and formerly RSA) is a comprehensive design, modeling and development tool for end-to-end software delivery URL: http://www-03.ibm.com/software/products/en/ratsadesigner
4. XML Metadata Interchange (XMI) Specification/ Version 2.5.1. URL: http://www.omg.org/spec/XMI/2.5.1/PDF.
5. ArgoUML, leading open source UML modeling tool URL: http://argouml.tigris.org/.
6. Sparx Systems Enterprise Architect is a visual modeling and design tool based on the OMG UML. URL: http://www.sparxsystems.com.au/.

References

1. OMG Unified Modelling Language (OMG UML). Version 2.5. URL: http://www.omg.org/spec/UML/2.5/PDF.
2. StarUML is a UML tool by MKLab. URL: http://staruml.io/.
3. IBM® Rational® Software Architect Designer (RSAD and formerly RSA) is a comprehensive design, modeling and development tool for end-to-end software delivery URL: http://www-03.ibm.com/software/products/en/ratsadesigner.
4. XML Metadata Interchange (XMI) Specification/ Version 2.5.1. URL: http://www.omg.org/spec/XMI/2.5.1/PDF.
5. ArgoUML, leading open source UML modeling tool URL: http://argouml.tigris.org/.
6. Sparx Systems Enterprise Architect is a visual modeling and design tool based on the OMG UML. URL: http://www.sparxsystems.com.au/.

Поступила 12.10.2016

Об авторах:

Ерохин Максим Владимирович, магистрант Московского авиационного института (национального исследовательского университета), xan.true@gmail.com.