Моделирование свободно-распространяемого программного обеспечения с помощью языка UML

Романов В.Ю.

***Аннотация* — В статье рассматриваются особенности CASE- инструмента для моделирования и визуализации программного обеспечения хранимого в репозиториях программного кода. Описаны особенности реализации метамодели языка UML 2.4 на основе хранилища данных (Data Store) предоставляемого фирмой Google в своей инфраструктуре для разработки облачных приложений. Как и другие приложения, CASE-инструменты могут быть разработаны как облачные приложения предоставляющие сервис через интернет браузер. Данная реализация метамодели языка UML разработана для такого CASE-инструмента. При реализации метамодели использовался подход генеративного программирования по компактной спецификации модели. Для визуализации архитектуры программной системы используется стандарт ISO/IEEE на описание архитектуры программной системы.**

***Ключевые слова* — artifact repository, Google App Engine datastore, metamodel implementation, software architecture, UML.**

I. Введение

В последнее время при разработке программного обеспечения на языке Java все большее популярность получает использование свободно распространяемых в сети Internet библиотек. Многие такие библиотеки распространяются и в исходных текстах на языке Java. Разработка программного обеспечения в таком случае представляет собой сборку программного обеспечения с указанием уникального идентификатора группы (фирмы или команды разработчиков), уникального идентификатора артефакта (библиотеки) и номера версии артефакта. Заданная в проекте с помощью таких координат библиотека автоматически подгружается из глобального репозитория в интернете в локальный репозиторий проекта и используется при сборке программы. Требуемые загруженной библиотекой другие библиотеки также рекурсивно подгружаются в локальный репозиторий и используются при сборке вновь разрабатываемого программного обеспечения на языке Java. Широкое распространение получили также инструменты [1, 2] позволяющие автоматизировать с помощью координат описание зависимостей между библиотеками и их сборку. Разработана унифицированная структура репозиториев для хранения таких библиотек [3] и программный интерфейс [4] для работы с репозиториями артефактов, которые имеют стандартизованную структуру.

Постоянное заполнение репозиториев новыми различными или аналогичными артефактами, наличие многочисленных версий таких артефактов, которые зачастую развиваются параллельно разными группами разработчиков в собственном направлении, существенно повышает роль человеческого фактора при использовании таких репозиториев. Понимание назначения и структуры таких библиотек-артефактов, их сходств и отличий требует значительных усилий и времени при их выборе и использовании. Это делает актуальным построение и визуализацию моделей артефактов, их внутренней структуры и связей между ними. Весьма актуальной становится задача обратного проектирования (reverse engineering) программного обеспечения с помощью языка моделирования UML. Для построения UML модели программного обеспечения репозитория используются предоставляемые репозиторием описания зависимостей между артефактами, описания зависимостей между компонентами сделанными в соответствии со спецификацией OSGi[5] и извлекаемой из исполняемых jar-файлов, анализ структуры байткода виртуальной машины Java.

В стандарте на язык UML версии 2.4 [6], в частности, содержится описание модели самого языка UML называемой метамоделью языка UML. Описание метамодели содержится описание классов, из экземпляров которых строится UML-модель, а также описывается графическая нотация для представления классов метамодели и их связей на UML-диаграммах. Для такого описания классов метамодели используется графическая нотация языка UML. Для формализованного представления графической нотации языка UML также стандарте определены классы, из экземпляров которых должно строиться изображение UML-диаграмм [7]. Для обмена моделями и диаграммами между поддерживающими стандарт CASE инструментами в стандарте определено отображение классов метамодели в расширение языка XML – формат XML Metadata Interchange (XMI) [8].

Все большее распространение получают приложения, разработанные как облачные приложения доступные

Владимир Юрьевич Романов, факультет ВМК МГУ им. М.В. Ломоносова, vromanov@cs.msu.su, vladimir.romanov@gmail.com.

пользователю через обычный Web-браузер. Использование таких приложений избавляет пользователя от скачивания приложения на свой компьютер, обновления его версий, и позволяет использовать приложение только при необходимости. В частности, такими облачными приложениями могут быть и CASE-инструменты для моделирования и визуализации программного обеспечения хранимого в репозиториях свободного программного кода. Широкое распространение получили облачные приложения, работающие в инфраструктуре фирмы Google и разработанные с помощью пакета Google App Engine[9]. Вместе с тем для разработки CASE-инструмента для этой, в целом, удобной инфраструктуры, необходима собственная реализация метамодели языка UML. Это обусловлено запретом использования облачными приложениями Google файловой системы для записи информации. Использование получившей широкое распространение реализации метамодели UML в составе платформы Eclipse[10] требует, в силу упомянутой причины, существенных переделок кода реализации, а также затруднено привязкой этой реализации к платформе Eclipse. Поэтому в CASE-инструменте применяется собственная реализация метамодели, которая использует для записи информации о модели хранилище данных Datastore[11] инфраструктуры Google.

II. Разработка метамодели языка UML на основе ее формализованной спецификации

Описание языка UML состоит из нескольких объемных документов превышающих [6, 7, 8] 700 страниц, которые постоянно изменяются и дополняются при появлении новых версий стандарта. Объем и регулярное изменение документов стандарта существенно затрудняют реализацию классов метамодели и взаимодействующих с ней компонентом метамодели. Для упрощения реализации метамодели и поддержки соответствия реализации текущим версиям языка использовался подход генеративного программирования. Реализация метамодели написана на языке программирования Java не вручную, а сгенерирована из компактной спецификации метамодели. Генеративный подход позволяет избавиться от большого объема рутинной работы, избежать ошибок возникающих при непосредственном ручном написании программ, дает возможность опробовать множество различных подходов к реализации метамодели.

Выбор языка для создания компактной спецификации весьма важен. В документах, описывающих стандарт языка UML, классы метамодели и связи между ними описываются с помощью диаграмм UML с комментариями к диаграммам на английском языке. В дополнение к этим диаграммам в документе UML содержится описания на формальном текстовом языке Object Constraint Language (OCL)[12]. Такие описания содержат либо логические выражения, которые должны быть всегда истинными для атрибутов и ролей ассоциаций классов модели. Либо на языке OCL описываются алгоритмы для вычисления значений порождаемых атрибутов классов метамодели. Для формального описания классов метамодели в дополнение к текстам, которые описывают стандарт, предоставляется также и формализованное описание модели на расширении языка XML – языке XMI. Вместе с тем такое описание на языке XMI предоставляется лишь для уже официально принятых версий стандарта и отсутствует для промежуточных версий.

Наличие этих трех разрозненных и существенно различающихся языков описания стандарта UML затрудняет их использование для генерации реализации метамодели на языке программирования. Удобная для визуального восприятия графическая нотация затрудняет ввод информации о метамодели в генерирующую программу. Для текстового описания на языке OCL нет соответствующего текстового описания контекста, в котором выражения на OCL должны вычисляться. Описание на языке XMI удобно для распознавания программой, но затрудняет определение его соответствия описанию на языке графической нотации, а также определения контекста для вычисления выражений на языке OCL.

По указанным выше причинам в качестве языка спецификации был разработан и использован текстовый язык TinyUML, синтаксис и семантика которого эквивалентны используемому для описания классов метамодели подмножеству графической нотации. Нотация языка TinyUML приближена к обозначениям, используемым на UML-диаграммах и описании стандарта. Текстовая природа языка позволяет в одном текстовом файле хранить описание классов и связей между ними, которое также является контекстом для вычисления выражений на языке OCL, а также служить исходными данными для генератора реализации на языке программирования Java.

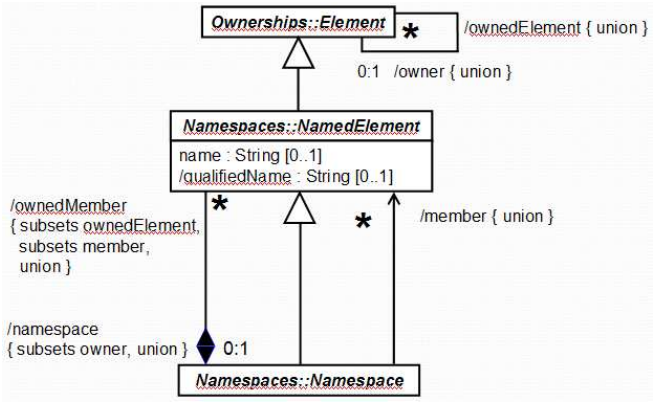

Рис. 1. Фрагмент спецификации метамодели языка UML с использованием графической нотации UML

Эквивалентный приведенному на рисунке фрагменту спецификации метамодели текст на языке TinyUML приведен ниже.

```
package Ownership {

abstract class Element {};

  as /ownedElement[*] : Element {union}
      <->/owner[0:1] : Element {union};
};

package Namespace {
  abstract class NamedElement :
                      Ownership::Element
  {
    at name[0:1] : String = "";
    at /qualifiedName[0:1] = "";
  };

  abstract class Namespace : NamedElement {};

  as /member[*] : NamedElement {union}
       <- ns[1] : Namespace;

  as /ownedMember[*] : NamedElement {union,
       subsets ownedElement, subsets member}
        <-> /namespace[0:1] : Namespace
        {union, composite, subsets owner};
};
```

Как можно заметить, текстовые элементы, показанные на UML диаграмме, в языке TinyUML полностью повторяются. Графические элементы нотации UML представлены в виде текста. Так, например, отношение ассоциации между классами представлено с помощью ключевого слова as языка TinyUML, а направленность отношения ассоциации – символами больше и равно.

На приведенном выше фрагменте спецификации описаны базовые классы метамодели, представляющие наиболее фундаментальные свойства элементов модели. Так, из спецификации следует, в частности, что пространство имен (представлено абстрактным классом Namespace) может включать в себя неограниченно число именованных элементов (представленных абстрактным классом NamedElement), которые могут входить не более чем в одно пространство имен. Указанные классы служат базовыми классами для конкретных классов, которые представляют в модели пространства имен и их элементы. Например, для пакетов, классов и интерфейсов представляющих в метамодели соответствующие конструкции объектно-ориентированных языков программирования.

III. Реализация метамодели языка UML как части облачного приложения фирмы Google

В последнее время все большее распространение получает разработка приложений как веб-сервисов расположенных на облаке и доступных через обычный интернет браузер. Как такое облачное приложение разработан и описываемый CASE-инструмент. Инфраструктура фирмы Google[9] для разработки облачных приложений получает все большее распространение. Для разработки как клиентской, так и серверной части предоставляет инструментарий для написания приложения, в частности, на языке Java.

Особенностью облачных приложений фирмы Google является способ хранения данных этим приложением на сервере. Файловая система сервера, на котором расположено приложение, может использоваться приложением лишь для хранения статической информации. Для работы же с изменяемой информацией инфраструктура облачного приложения предоставляет специализированное хранилище данных Data Store [11]. Хранилище данных представляет собой масштабируемую объектную базу данных, каждый объект (Entity) которой может включать в себя одно или несколько свойств. Свойства объектов имеют имена и могут хранить значения одного из поддерживаемых типов данных, а также могут быть ссылкой на другой объект. Для моделирования отношения ассоциации между классами модели используются описание связей между сущностями хранилища данных [13]. Это позволяет создавать между объектами, в частности, отношения один-к-многим и многие-к-многим. Некоторые из свойств могут быть ключами объекта и использоваться для уникальной идентификации объектов. Для работы с данными хранилище данных предоставляется схожий с языком SQL язык запросов Google Query Language (GQL) [14]. Механизм транзакций хранилища данных используется для реализации изменения состояния базы данных сгенерированными методами классов метамодели.

Указанные возможности хранилища данных была использованы для реализации метамодели языка UML как множества хранимых в базе специализированных объектов-сущностей. Реализация метамодели генерировалась на языке Java по спецификации метамодели. Базовым классом такой реализации классов метамодели является класс-сущность из API хранилища данных. Таким образом, от базового класса из API хранилища данных классами метамодели наследуется возможность чтения и записи свойств сущности, которые представляют атрибуты и отношения ассоциации определенные в стандарте для классов метамодели. Используемые для уникальной идентификации ключи сущностей хранилища используются для реализации отношений ассоциации определенных между классами метамодели. Для реализации порождаемых атрибутов-объединений (union) у классов метамодели, значения которых вычисляются как объединения подмножеств значений атрибутов других классов, используются запросы на языке GQL.

Стандарт на язык UML[6, 7, 8] не содержит явно описания функционального интерфейса предоставляемого классами метамодели. Вместе с тем свойства и множественность атрибутов и ролей отношения ассоциации неявно определяют имена и сигнатуры методов классов метамодели. Разработавшим стандарт языка UML консорциумом фирм Object Management Group разработан язык определения интерфейсов IDL[15] используемый при описании функциональных интерфейсов для разрабатываемых

консорциумом стандартов. Вместе с тем для одного из подмножеств языка UML отображение классов метамодели в спецификацию на языке IDL[16] определено явно. Определенная аналогичным образом функциональность классов метамодели поддерживается и в широко используемой реализации метамодели UML для среды Eclipse[10]. Такой же функциональный интерфейс определен и в реализации метамодели использующей хранилище данных Google.

IV. Моделирование архитектуры программного обеспечения

Графическая нотация языка UML позволяет представить на одной диаграмме в компактной форме информацию, которая зачастую рассредоточена в нескольких текстовых файлах написанных на объектно-ориентированном языке программирования. Вместе с тем при большом размере программного кода, хранимого в репозиториях, полная его визуализация с помощью языка UML, может оказаться затруднительной. Ее восприятие по-прежнему будет затруднено из-за большого числа UML-диаграмм. Необходима визуализация наиболее существенной информации о программной системе – ее архитектуре.

В результате совместной работы международных организаций по стандартизации ISO и IEEE был принят стандарт, определяющий, что может считаться описанием архитектуры программного обеспечения [17]. Были также изданы книги [18,19] являющиеся хорошими комментариями к данному стандарту. В стандарте и упомянутых книгах формализованным образом определяются элементы, из которых строится описание архитектуры (architectural elements), категории участников заинтересованных в реализации программной системы (stakeholders). Для каждой категории участников из элементов строится описание архитектуры. В стандарте определен также каталог проекций (views) и точек зрения на модель (viewpoint) с подмножествами типов элементов архитектуры, из которых они строятся. В стандарте регламентируется деятельность системного архитектора при построении им описания архитектуры. Частично работа архитектора по построению модели архитектуры программного обеспечения (проекций и различных точек зрения) и ее визуализации, выполняемая для хранимого в репозиториях свободного программного кода, может быть автоматизирована CASE-инструментом.

V. Заключение

Рассмотрены особенности CASE-инструмента для моделирования и визуализации программного обеспечения хранимого в репозиториях программного кода. Важнейшей компонентой CASE-инструмента является репозиторий для хранения моделей программных систем. Определение в стандарте на язык UML [6, 7, 8, 12] модели репозитория CASE-инструмента, выполненное с помощью графической нотации самого языка UML, получило название метамодели языка UML. Большой объем стандарта (более 700 страниц) существенно осложняет непосредственную реализацию метамодели. В статье рассматривается применение апробированного ранее при реализации на языке C# метамодели для CASE-инструмента интегрированного в среду Visual Studio[20]. Данный подход был успешно применен и к программной генерации реализации метамодели на языке Java с использованием хранилища данных фирмы Google. Данная реализация метамодели разработана для CASE-инструмента [21] реализованного как облачное приложение фирмы Google. При визуализации архитектуры программной системы CASE-инструментом используется стандарт ISO/IEEE на описание архитектуры программной системы.

Библиография

[1] Apache Maven, http://maven.apache.org/
[2] Ivy - The agile dependency manager, http://ant.apache.org/ivy/
[3] Maven architecture, http://maven.apache.org/ref/3.0.5/
[4] Aether- the library for working with artifact repositories, http://www.eclipse.org/aether/
[5] OSGi - The Dynamic Module System for Java, http://www.osgi.org
[6] Object Management Group, UML 2.4 Superstructure Specification, OMG document. http://www.omg.org/spec/UML/2.4.1/
[7] Object Management Group, UML Diagram Interchange, OMG document, http://www.omg.org/spec/UMLDI/1.0/PDF
[8] Object Management Group, XML Metadata Interchange, OMG document, http://www.omg.org/spec/XMI/2.4.1/
[9] Разработка и выполнение облачных приложений в инфраструктуре фирмы Google. http://code.google.com/intl/ru-RU/appengine/
[10] EMF-based implementation of the UML 2.x OMG metamodel for the Eclipse platform. http://www.eclipse.org/modeling/mdt/?project=uml2
[11] Хранилище данных Data Store App Engine фирмы Google. http://www.googleappengine.ru/docs/datastore/
[12] Object Constraint Language, OMG document, http://www.omg.org/spec/OCL/2.3.1/
[13] Modeling Entity Relationships, https://developers.google.com/appengine/articles/modeling#related
[14] Описание языка Google Query Language. http://www.googleappengine.ru/docs/datastoregqlrefere/nce.html
[15] Interface Definition Language (IDL) 3.5 http://www.omg.org/spec/IDL35/
[16] MOF to IDL Mapping, http://www.omg.org/spec/MOF2I/2.0/
[17] ISO/IEC/IEEE 42010:2011, Systems and software engineering — Architecture description, http://www.iso-architecture.org/ieee-1471/
[18] Nick Rozanski, Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives, Second Edition, Addison-Wesley Professional, October 25, 2011, ISBN 978-0-321-71833-4
[19] Paul Clements; Felix Bachmann; Len Bass; David Garlan; James Ivers; Reed Little; Paulo Merson; Robert Nord; Judith Stafford, Documenting Software Architectures: Views and Beyond, Second Edition Publisher: Addison-Wesley Professional, October 05, 2010, ISBN 978-0-321-55268-6
[20] В.Ю.Романов, Реализация метамодели языка UML 2.0 на языке C#. Сборник трудов первой международной научно-практической конференции “Современные информационные технологии и ИТ-образование». с.332-339, 19-23 сентября 2005 г.
[21] Романов В.Ю. Сервис анализа и визуализации кода и текстов на языках программирования как облачное приложение Google. Сб. трудов V Международной научно-практической конференции "Современные информационные технологии и ИТ-образование". М.:, 2011. с.743-748, 12-14 декабря 2011 г.

Free software modeling with UML language

Romanov V.Y.

***Abstract* — The article describes CASE-tool for modelling and visualization of software code from repositories. UML 2.4 metamodel implementation based on Google datastore is described. The metamodel implementation used in CASE-tool that is Google cloud application. The metamodel code was generated from compact metamodel specification. The ISO/IEEE 42010 standard is used for modeling and visualization of software architecture.**

***Keywords* — artifact repository, Google App Engine datastore, metamodel implementation, software architecture, UML.**