МАТЕМАТИЧЕСКИЕ МОДЕЛИ
В ИНФОРМАТИКЕ
(Вычислительные системы)
2002 год Выпуск 169
УДК 519.682.2:519.682.6:519.685.1

АВТОМАТИЗАЦИЯ ПРОЦЕССОВ РАЗРАБОТКИ НА ОСНОВЕ RATIONAL UNIFIED PROCESS

О.Ю. Чунихин

Введение

Данная работа посвящена обзору современных технологий и продуктов построения больших программных систем на основе объектно–ориентированного подхода и, в частности, семейства стандартов фирмы Rational, включающих Unified Modelling Language и Rational Unified Process. В работе рассматриваются как продукты уже представленные на рынке и завоевавшие признание разработчиков, так и новые разработки.

Первая часть статьи кратко описывает основные понятия упомянутых технологий и стандартов: объектно–ориентированного программирования, UML и RUP.

Объектно–ориентированный подход — не новая концепция. Первые понятия объектно–ориентированного подхода возникли и развились в конце 1960 г. в языке Simula67. До настоящего времени объектно–ориентированный подход развивался параллельно с развитием вычислительной техники и технологий. Объектно–ориентированный подход позволяет лучше управлять технологической сложностью, что дает возможность разрабатывать системы на все более высоких уровнях абстракции — от объектов к классам, затем к библиотекам классов, и в итоге к наборам инструментов создания приложений. Объектно–ориентированный подход увеличивает степень переиспользования кода, программы гораздо проще адаптируются к меняющимся обстоятельствам. Еще одно преимущество объектно–ориентированного подхода — более последовательная модель процесса разработки программного обеспечения.

Другой стороной развития объектно-ориентированного подхода стало развитие техник моделирования программных (и не только программных) систем. Предшественники объектно-ориентированных языков моделирования начали появляться в середине 1970 г., и продолжали возникать в течение 1980 г., по мере того, как программисты и методологи пробовали различные подходы к объектно–ориентированному анализу и дизайну. В июне 1996 г., как итог многолетней работы и сумма предыдущего опыта, была опубликована спецификация UML (Unified Modeling Language) версии 0.9. UML вызвал большой интерес и привлек множество организаций. Они присоединились к консорциуму UML партнеров, организованному Rational Software; среди этих организаций были такие компании, как HP, Microsoft, Oracle, Unisys и IBM. В результате взаимодействия этих компаний появилась версия UML 1.0, строго определенный, выразительный, мощный и широко применимый язык моделирования. Сейчас UML широко применяется в самых различных областях, от программирования до бизнес–анализа, для моделирования различных систем.

При создании большой программной (и не только программной) системы имеет значение не только общий язык, на котором описывается система и ее части, но и грамотная организация работ, взаимодействие людей, производящих эти работы. Обычно организация работ описывается в терминах процессов, в данном конкретном случае бизнес–процессов. Для того, что бы не приходилось каждый раз разрабатывать эти процессы с нуля, корпорацией Rational Software и ее идеологами Гради Бучем, Айваром Джекобсоном и Джеймсом Рамбо, был создан RUP — Rational Unified Process — набор проверенных на практике принципов, методов и процессов качественной и производительной работы над проектами по созданию программного обеспечения. RUP в терминах UML описывает стандартный набор ролей, бизнес–процессов, артефактов и т.д., необходимых при разработке программного обеспечения.

Вторая часть статьи посвящена классификации и описанию продуктов, позволяющих организовать и частично автоматизировать процесс разработки программного обеспечения. Выделены следующие основные группы подобных продуктов: UML редак-

торы или средства моделирования, генераторы, интегрированные средства и метаинструменты.

§ 1. Технологии разработки программного обеспечения

1. Объектно–ориентированный подход. Первые понятия объектно–ориентированного подхода возникли и развились в конце 1960 г. в языке Simula67, разработанном Кристином Нигаардом (Kristen Nygaard) и Оле–Иоганном Далом (Ole–Johan Dahl) в Норвежском Вычислительном Центре. Именно в Simula67 впервые возникли концепции класса, подкласса и процедуры класса. В середине 1970 г. в исследовательском центре Пало Альто компании Ксерокс был создан язык Smalltalk — первый полный и ясный объектно–ориентированный язык. В Smalltalk все элементы языка были реализованы как объекты, программное окружение и культура программирования соответствовали философии объектно–ориентированного подхода. Даже сейчас Smalltalk считается наиболее чистым объектно–ориентированным языком.

Simula и Smalltalk развили новый стиль программирования, когда данные и процедуры их обработки хранятся вместе, что существенно отличалось от традиционного процедурного подхода (реализованного в таких языках, как C, Pascal, Fortran). Но хотя Simula и Smalltalk были широко известны в академических кругах, они были относительно не востребованы основным большинством программистов и объектно–ориентированный подход не находил широкого применения в коммерческих системах. В начале 1980 г. Бьярн Страуструп (Bjarne Stroustrup), работавший в AT&T Bell Laboratories создал язык C++ — объектно–ориентированное расширение языка C. Объектно–ориентированный подход получил широкое распространение именно после коммерческого выпуска C++. Уже в конце 1980 г. объектно–ориентированное программирование стало практически стандартом в индустрии информационных технологий. Несмотря на то, что процедурно–ориентированный подход имеет свои преимущества в специфических проектах, в общем можно сказать, что объектно–ориентированный подход является в настоящее время наилучшей методологией разработки ориентированных на поль-

зователя интерактивных программных систем, и программных систем средней и высокой сложности.

В настоящее время от разработчиков приложений требуется удовлетворение все более и более сложных требований, использование все более сложных структур данных и архитектур, предложение сервисов все большему количеству конечных пользователей. Эти факторы заставляют разработчиков строить все более сложные и масштабные системы. Объектно–ориентированный подход позволяет лучше управлять технологической сложностью. Он дает возможность разрабатывать системы на все более высоких уровнях абстракции — от объектов к классам, затем к библиотекам классов, и в итоге к наборам инструментов создания приложений. Объектно–ориентированный подход увеличивает степень переиспользования кода, программы гораздо проще адаптируются к меняющимся обстоятельствам. Еще одно преимущество объектно–ориентированного подхода более последовательная модель процесса разработки программного обеспечения.

Рассмотрим основные понятия объектно–ориентированного подхода.

Основной идеей объектно–ориентированного подхода является инкапсуляция данных вместе с методами обработки этих данных в сущности, называемые объектами. Методы при этом подходе описывают поведение объектов. Данные представлены атрибутами объекта, говорят, что атрибуты объекта описывают его состояние.

Объекты, ведущие себя одинаково, и чье состояние описывается одинаковыми наборами атрибутов, группируются в классы. Класс, в этом смысле, является описанием группы объектов.

Основными чертами объектно–ориентированного подхода являются инкапсуляция, передача сообщений, наследование и полиморфизм.

• Инкапсуляция — это методика сокрытия от пользователя деталей реализации путем четкого разделения внутренней структуры объекта и внешнего его представления. Это делает программу нечувствительной к небольшим изменениям реализации. Инкапсуляция также подразумевает комбинацию описания

структуры данных и поведения объектов и классов, что делает систему более прозрачной. Например, в объектно-ориентированной системе может быть описан класс Currency, представляющий денежную сумму. На нем определены методы, позволяющие увеличить — add(Currency) — и уменьшить сумму — subtract(Currency). Если система правильно спроектирована, то смена валюты в стране (например, переход от расчетов в рублях к расчетам в рублях и копейках) не затронет в системе ничего, кроме реализации класса Currency (в данном случае внутреннее представление Currency целым числом будет заменено представлением вещественным числом).

• Передача сообщений — единственный метод взаимодействия объектов в объектно-ориентированном подходе. Вызов функции-члена объекта, или метода в терминологии объектно-ориентированного подхода, является передачей сообщения объекту. В вышеприведенном примере вызов метода add на объекте класса Currency можно воспринимать как посылку объекту класса Currency сообщения add с параметром — другим объектом класса Currency, что приведет к увеличению суммы, представленной первым объектом на сумму, представленную вторым.

• Наследование позволяет разрабатывать иерархические системы классов, начиная с самых общих определений, и заканчивая самыми специфическими определениями. Другими словами, это дает возможность переиспользовать существующие определения при создании новых. Новые классы можно считать описаниями подмножеств объектов, определенных старыми классами. Кроме того, это дает возможность разделять данные и функциональность между связанными объектами, избегая, таким образом, повторной реализации одних и тех же функций. К примеру, в системе может быть определен класс Person, описывающий работника организации. Класс может иметь общие для всех работников атрибуты (дата-рождения, дата-приема-на-работу, зарплата) и методы (принять-на-работу, уволить, начислить-зарплату). Кроме того, на предприятии есть директор, который, с одной стороны, является обычным работником, но с другой стороны, обладает специфическими свойствами-атрибутами (список-подчиненных) и функциями-методами (нанять-нового-

работника). Можно было бы продублировать в классе Director все атрибуты и методы класса Person, но проще и правильнее просто объявить класс Director наследником класса Person, и описать в нем только специфические атрибуты и методы.

• Полиморфизм дает возможность создавать структуры, которые можно использовать с различными, хотя и похожими типами. С его помощью дизайнер может создавать общие методы, работающие с суперклассами, которые могут так же работать и с субклассами, что делает код более гибким и расширяемым. Система автоматически применяет нужный метод, используя информацию о типе. Так, в предыдущем примере начисление зарплаты может происходить по разным алгоритмам для обычного работника и директора предприятия (работнику начисляется фиксированная зарплата, а директору — зависящая от общей прибыли предприятия). В этом случае в классе Director будет соответствующим образом переопределен метод начислить-зарплату, а сам процесс начисления будет выглядеть просто как вызов этого метода на всех объектах класса Person, в том числе и на объекте класса Director.

2. Unified Modeling Language. Другой стороной развития объектно-ориентированного подхода стало развитие техник моделирования программных (и не только программных) систем. Предшественники объектно-ориентированных языков моделирования начали появляться в середине 1970 г., и продолжали возникать в течение 1980 г., по мере того, как программисты и методологи пробовали различные подходы к объектно-ориентированному анализу и дизайну. С 1989 по 1994 годы количество языков моделирования возросло с менее чем десяти до более чем пятидесяти. Такое богатство выбора означало, что множество пользователей объектно-ориентированных техник моделирования не могли выбрать язык, наиболее полно удовлетворяющий их потребности. В середине 1990 г. возникли новые объектно-ориентированные техники моделирования, инкорпорирующие в себе возможности и черты большого количества предшественников; этими техниками стали Booch'93 (по имени создателя Гради Буча) [1], OMT-2 (object modeling technique) [4,5] и OOSE (object-oriented software engineering, создана в шведской

компании "Objectory") [6,7]. Каждая из них имела свои сильные стороны: OOSE был ориентирован на "случаи использования" и очень хорошо подходил для разработки и моделирования бизнес-процессов, OMT-2 позволял легко проводить анализ и разработку информационных систем со сложными и интенсивными потоками данных, Booch'93 очень хорошо подходил для фаз дизайна и реализации процесса разработки программного обеспечения.

В октябре 1994 г. Гради Буч (Grady Booch) и Джим Румбаух (Jim Rumbaugh), работавшие в Rational Software Corporation, начали унификацию Booch и OMT и в октябре 1995 г. представили предварительный вариант техники "Unified Method".

В то же самое время прилагались значительные усилия по выработке промышленного стандарта на рынке языков моделирования. В начале 1995 г. года Айвар Джекобсон (Ivar Jacobson), тогда главный технический руководитель Objectory и Ричард Солей (Richard Soley), тогда главный технический руководитель OMG (object management group), договорились о совместной деятельности в области стандартизации на рынке ОО техник. В июне 1995 г. на встрече главных методологов в OMG было выработано первое широкомасштабное соглашение о разработке методологического стандарта.

OOSE была включена в United Method когда компания "Objectory", разработавшая его, была куплена Rational Software. Ее предыдущий владелец, Айвар Джекобсон, присоединился к Бучу и Румбауху в их усилиях, направленных на унификацию (в дальнейшем эта тройка получила прозвище "Amigos"). Основными мотивами создания унифицированного языка моделирования (Unified Modeling Language — UML) были:

• очевидно, три уже существующие техники независимо развивались навстречу друг другу, и имело смысл избавиться от несущественных и мешающих различий;

• объединение трех техник в одну стабилизировало бы рынок объектно-ориентированных методологий и позволило бы разработчикам сконцентрироваться на доработке элементов уже существующего языка, а не распылять силы на разработку новых языков;

• предполагалось, что объединенная техника не будет иметь недостатков предшественников и будет объединять их положительные стороны.

Были поставлены следующие цели:

1) язык должен позволять моделировать системы (и не только программные) используя объектно–ориентированные концепции;

2) устанавливать явные зависимости как между концептуальными, так и между исполняемыми артефактами (объектами процесса разработки);

3) работать с проблемами масштаба, присущими сложным системам;

4) язык должен использоваться как человеком, так и машинами.

В июне 1996 г. Буч, Румбаух и Якобсон опубликовали спецификацию UML 0.9 и пригласили сообщество программистов и методологов к участию в обсуждении языка.

UML вызвал большой интерес и привлек множество организаций. Они присоединились к консорциуму UML партнеров, организованному Rational Software; среди этих организаций были такие компании, как HP, Microsoft, Oracle, Unisys и IBM. В результате взаимодействия этих компаний появилась версия UML 1.0, строго определенный, выразительный, мощный и широко применимый язык моделирования. Этот стандарт был представлен OMG в январе 1997 г. В то же время дополнения, представленные IBM, ObjecTime, Platinum Technology, Ptech, Taskon, Reich Technologies и Softeam, были включены в стандарт, что позволило выпустить версию UML 1.1, в которой была более ясно определена семантика. Эти дополнения более точно описывали такие аспекты языка, как бизнес–моделирование, язык описания ограничений, семантику машин состояний, типы, интерфейсы, компоненты, диаграммы взаимодействия и метамодели. Стандарт UML 1.1 был принят 17 ноября 1997 г. Текущая версия UML — UML 1.4 [3,11,12,14].[1] Кроме того, OMG разработал

[1] Смотри http://www.omg.org/technology/uml/index.htm.

XML (eXtensible Markup Language) стандарт на представление UML моделей - XMI (XML Metadata Interchange),[2]

В архитектуре UML важнейшим элементом является понятие метамодели — модели более высокого уровня. Так, можно считать, что конкретная система объектов является моделью самой себя нулевого уровня. Модель, описывающая классы этих объектов, их ассоциации и поведение является моделью первого уровня и называется просто моделью. Ее структура обычно проще, чем структура системы. Модель, описывающая классы элементов различных моделей первого уровня (классы, ассоциации, атрибуты, операции, состояния и т.д.) является моделью второго уровня или метамоделью. Кроме того, в архитектуре UML определяется наличие метаметамодели, или модели третьего уровня, описывающей структуру различных метамоделей. Ее структура уже очень проста и не поддается дальнейшему упрощению (т.е. модель четвертого уровня, или метаметаметамодель, будет иметь ту же структуру, что и модель третьего уровня, или метаметамодель). Модели всех уровней описаны средствами UML в спецификации языка.

Типичные объекты, входящие в статическую UML модель — это классы и отношения между ними. Классы — это самые важные строительные блоки в любой объектно-ориентированной системе. Они представляют собой описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой. Класс может реализовать один или несколько интерфейсов.

Кроме того, очень важным свойством UML является его расширяемость. UML задумывался как язык моделирования, применимый в самых различных областях, и, поскольку на этапе проектирования невозможно было предусмотреть все области применимости UML, в язык были заложены механизмы расширения. Важнейшим из них являются стереотипы. Стереотип — метка, придающая элементу UML специфический смысл. Так, например, UML-класс, помеченный стереотипом "boundary", обозначает интерфейсный класс, то есть класс объектов, обеспечивающих связь системы с внешним миром. Для каждого из

[2]Смотри http://www.omg.org/technology/xml/index.htm.

конкретных применений UML могут определяться свои наборы стереотипов.

Поскольку UML — язык моделирования, который должен быть понятен как человеку, так и машине, в нем существует понятие диаграммы. Если UML–модель может быть очень сложной и малодоступной пониманию одного человека целиком, то диаграмма всегда является некоторым "осознаваемым" срезом модели, который можно охватить одним взглядом. Обычно одной UML–модели соответствует множество UML–диаграмм, представляющих эту модель в виде, понятном человеку с различных точек зрения. Существует множество типов UML–диаграмм, каждый предназначен для какой–то определенной цели и может содержать определенный набор классов элементов модели:

- use–case диаграммы (диаграммы прецедентов) используются для выделения внешних по отношению к системе или процессу сущностей, и функций, предоставляемых системой этим сущностям;
- class диаграммы (диаграммы классов) описывают классы, из которых состоит система или ее части, все возможные связи между классами, полностью определяют множество состояний системы объектов;
- object диаграммы (диаграммы объектов) изображают систему объектов в какой–то определенный момент времени, дают представление о частном случае, мгновенном состоянии системы;
- на sequence диаграммах (диаграммах последовательностей) и collaboration диаграммах (диаграммах взаимодействий) отображается динамика взаимодействия объектов. С помощью этих диаграмм отображаются возможные сценарии взаимодействия объектов, последовательность обмена сообщениями, потоки управления и данных. Эти два типа диаграмм отличаются только внешним представлением, но несут одну и ту же семантику;
- что бы полностью определить динамику взаимодействия объектов, существуют state диаграммы (диаграммы состояний) и activity диаграммы (диаграммы активностей);
- логические компоненты программной системы отображаются на component диаграммах (компонентных диаграммах). Компоненты могут изображать как статические (пакеты, библио-

теки, исполняемые модули) и динамические (потоки, процессы, программы) элементы системы, так и ее физические элементы или узлы (сервер, компьютер, мобильный телефон и т.д.). Компоненты наследуют все свойства обычных UML–классов, т.е. так же могут иметь атрибуты, операции, интерфейсы, могут быть связаны друг с другом ассоциациями;

• на deployment диаграммах (диаграммах развертывания) отображаются возможные конфигурации системы, а именно, возможное распределение компонент системы по ее физическим узлам.

3. Rational Unified Process (RUP) — это процесс разработки программного обеспечения. Корпорация Rational Software и ее идеологи, столпы объектно–ориентированного подхода Grady Booch (Гради Буч), Ivar Jacobson (Айвар Джекобсон) и James Rumbaugh (Джеймс Рамбо), создали RUP–набор проверенных на практике принципов, методов и процессов качественной и производительной работы над проектами по созданию программного обеспечения. Физически RUP это база знаний, содержащая описания ролей всех участников процесса разработки, их активностей, ответственностей, шаблоны документов, которыми они обмениваются, стандарты на модели и большое количество другой необходимой информации [2,8,9,15].

Основными чертами RUP являются:

1) четкое разделение ролей разработчиков в рамках процесса, выделяются следующие группы разработчиков: аналитики, разработчики, тестировщики и менеджеры. В каждой из этих групп четко специфицируются отдельные роли;

2) выделяются следующие основные группы активностей: бизнес–моделирование, управление требованиями, анализ и дизайн (или проектирование), реализация, тестирование, передача, поддержка, управление проектом, управление изменениями и управление рабочим окружением;

3) есть следующие основные стадии разработки: запуск проекта (inception), разработка (elaboration), конструирование (construction) и передача (transition);

4) итеративность процесса (каждая стадия может быть разбита на несколько итераций, что позволяет более гибко управлять процессом);

5) определение исполнителей для каждой активности;

6) определение зависимостей активностей друг от друга;

7) общим языком анализа и проектирования является UML (Unified Modeling Language);

8) определяются общие стандарты моделирования на UML в рамках RUP.

По RUP процесс разработки программ представляет собой несколько основных процессов, идущих параллельно и постоянно активных. Кроме того, по времени он может быть разбит на фазы, которые, в свою очередь, могут быть разбиты на итерации.

На рис. 1 изображено примерное распределение активностей процессов по времени (в зависимости от фазы).

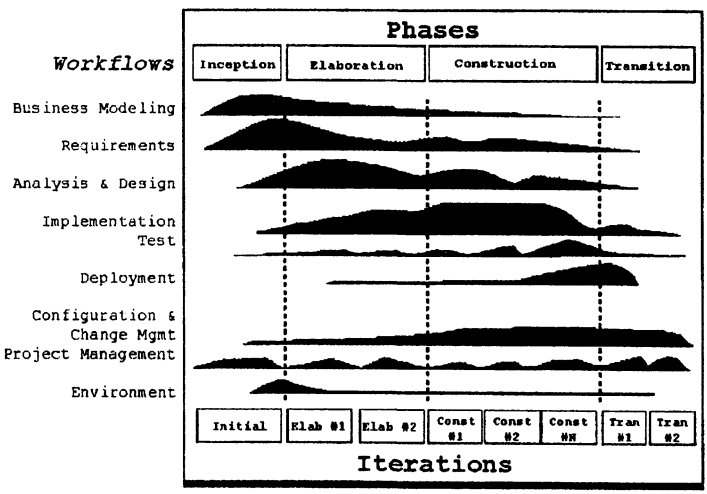

Рис. 1

Создание программного продукта начинается со стадии inception (начало). На этой стадии производится первоначальный

анализ запроса заказчика, разрабатывается предварительная архитектура системы. Наиболее важными активностями на этом этапе являются бизнес–моделирование, определение требований (управление требованиями), анализ и дизайн, конфигурация среды (environment).

Бизнес–моделирование — построения модели бизнеса, для автоматизации которого предназначена данная программная система. Бизнес–модель состоит из двух частей: моделей business use-case и business object. Первая описывает основные сценарии или процессы бизнеса, то, какие активности выполняются в рамках бизнеса. Вторая — структуру и взаимосвязи объектов и субъектов, участвующих в бизнесе, его предметную область и непосредственно описание того, каким образом совершаются действия, описанные в business use–case модели. Обе модели описываются на UML с помощью диаграмм объектов, use–case диаграмм, диаграмм состояний и активностей. Также бизнес–анализ выявляет те сценарии и объекты бизнеса, которые должны быть автоматизированы в результате внедрения программной системы.

Ниже приведены примеры business object (рис. 2) и business use–case–диаграмм (рис. 3).

Определение требований (или на следующих за стадией "inception" фазах управления требованиями) — активность, целью которых является разработка точной спецификации требований к системе и ее изменение в соответствии с меняющейся ситуацией. На этапе inception специфицируются функции системы (функциональные требования к системе или случаи использования системы — use–cases) и набор требований, не являющихся функциями системы, но являющихся существенными для возможности ее использования заказчиком (нефункциональные требования). Результатом этого этапа является документ, специфицирующий требования, которым должна удовлетворять готовая система — software requirements specification или SRS. Частью этого документа является модель случаев использования, описанная на UML с помощью use–case–диаграмм.

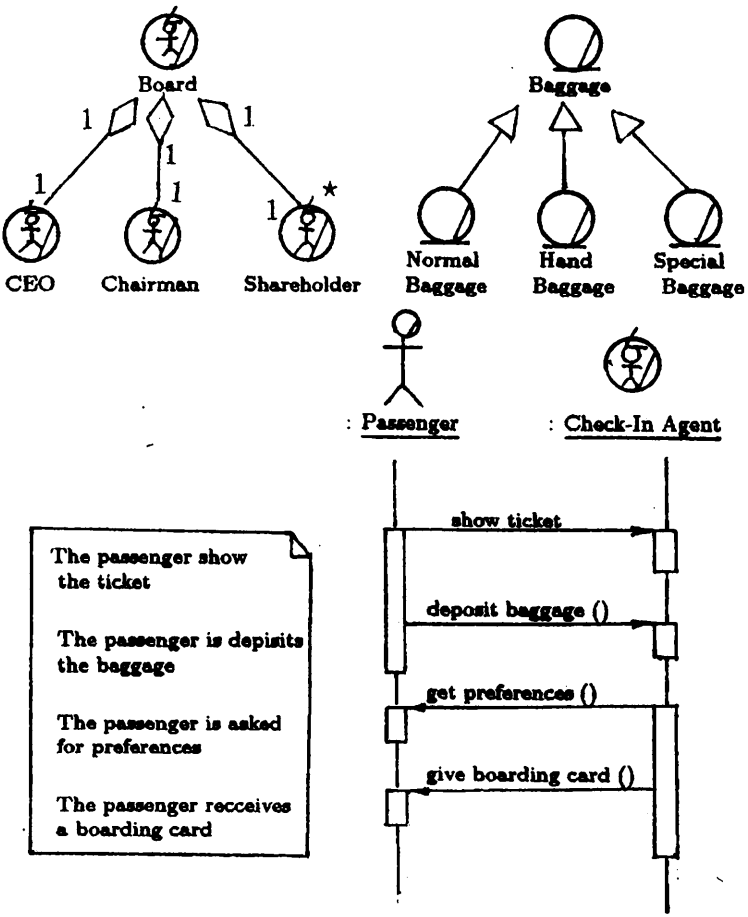
Board
1
1
1
1
1
1
*
CEO
Chairman
Shareholder
Baggage
Normal Baggage
Hand Baggage
Special Baggage
: Passenger
: Check-In Agent
The passenger show the ticket
The passenger is depisits the baggage
The passenger is asked for preferances
The passenger recceives a boarding card
show ticket
deposit baggage ()
get preferences ()
give boarding card ()

Рис. 2

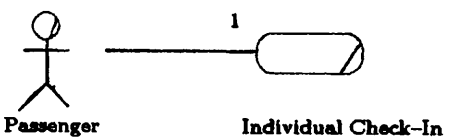
1
Passenger
Individual Check-In

Рис. 3

На рис.4 приведен пример use–case–диаграммы.

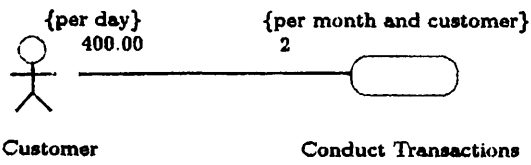

Рис. 4

Анализ и дизайн — это активность, цель которой — разработка такой архитектуры системы, которая полностью удовлетворяет всем требованиям, и изменение ее в соответствии с изменяющимися требованиями [10]. Архитектурная модель системы также специфицируется на языке UML и описывает черты внутреннего устройства системы, существенные для удовлетворения системой всех требований. Дизайн уточняет архитектурную модель, описывается внутренняя структура и особенности реализации всех модулей, описанных в архитектурной модели. При наличии достаточно точной дизайнерской модели становится возможной реализация системы.

Конфигурация среды — активность, целью которой является определение всех необходимых для разработки компонент — физических (компьютеров, другого оборудования) и программных (среды разработки, компиляторы, библиотеки, инструменты и т.д.).

Следующие две фазы — elaboration (разработка–уточнение) и construction (разработка) довольно похожи друг на друга. Цель и той и другой — создание работающего продукта. И та и другая могут быть разбиты на несколько итераций, результатом каждой и которых является предоставление заказчику промежуточного результата. Но основная цель первой фазы — уточнение требований к системе и гарантия того, что требования заказчика поняты верно; цель же второй — собственно разработка системы в условиях в основном зафиксированных требований. Иногда эти фазы сливаются в одну. Основными активностями на этих этапах являются анализ и дизайн, реализация и тестирование.

Цель реализации — создание отдельных программных элементов, составляющих систему. Тестирование предназначено для проверки функциональности системы и ее отдельных элементов, соответствия требованиям и выявления ошибок.

Последней фазой является transition (передача). На этом этапе готовая система передается заказчику, устанавливается, конфигурируется и поддерживается. Поддержка может быть как очень простой (например, помощь в установке программы), так и достаточно трудоемкой и продолжительной активностью (например, постоянный анализ изменений, происходящих в бизнесе клиента и адаптация системы в соответствии с меняющимися требованиями).

Кроме уже упомянутых активностей, есть еще активности, которые не имеют четкой локализации по фазам. Они существуют в течение жизни всего проекта; это управление требованиями и конфигурацией (configuration & change management) и управление проектом (project management). Первая подразумевает управление изменениями требований и связанными с этим изменениями архитектуры, дизайна и окружения проекта, вторая — решение возникающих организационных вопросов.

§ 2. Инструментарий разработчика и автоматизация процессов разработки

Существует огромное множество инструментов и продуктов, предназначенных для использования в процессах разработки, основанных на объектно–ориентированном подходе вообще и UML и RUP в частности. Конечно же невозможно рассмотреть все их в одной статье, но все же среди многообразия продуктов можно достаточно четко выделить классы и рассмотреть их представителей, наиболее характерных или популярных. Здесь выделены следующие основные группы подобных продуктов:

- UML редакторы или средства моделирования — программы, предназначенные для создания, редактирования и просмотра UML–моделей.
- Генераторы — инструменты, генерирующие различные артефакты процесса разработки частично или полностью по UML–модели системы.

• Интегрированные средства — системы разработки, включающие в себя как UML–редактор, так и другие инструменты. Чаще всего интегрированные средства разработки включают дополнительно один или несколько различных генераторов, часто — компиляторы и отладчики, иногда — интерпретаторы моделей.

• Метаинструменты — системы или модули, предназначенные для разработки инструментов, описанных выше.

1. Средства моделирования включают в себя UML–редакторы, но не ограничиваются ими. Дело в том, что существующие средства моделирования почти всегда включают в себя какие–либо средства генерации. По этой причине к данному классу инструментов отнесены не только "чистые" UML–редакторы, но и системы, поддерживающие "тривиальную" генерацию артефактов процесса разработки по UML–моделям.

Rational Rose Modeler [13] на сегодняшний день является наиболее известным и популярным инструментом моделирования UML. Это и неудивительно, если учесть то обстоятельство, что разработала Rational Rose фирма Rational Software Corp. — фирма, в которой работали создатели UML и которая, фактически, является законодателем в области основанных на UML–стандартов и технологий.

Основные черты продукта:

• поддержка прямого и обратного проектирования на языках: ADA, Java, C, C++, Basic. Поддерживает технологии COM, DDL, XML. Позволяет генерировать схемы Oracle и SQL;

• наличие открытого API, позволяющего создавать собственными силами модули для конкретных языков программирования. На рынке уже сейчас имеется достаточное число модулей для популярных языков программирования и систем, таких как Delphi, ErWin, Jbuilder, VisualCafe, Jdeveloper, VisualAge SmallTalk. Одна из ведущих компаний в области создания дополнительных модулей — Ensemble Systems;

• существует много шаблонов моделей, содержащих часто используемые элементы моделей для различных приложений, в том числе имеется и шаблоны моделей RUP;

• поддержка разбиения модели на несколько файлов, каждый из которых может редактироваться независимо от других;

• наличие встроенного скриптового языка, похожего на visual basic;

• наличие хорошей документации не только по продукту, но и по UML.

К сожалению, к минусам текущей версии продукта можно отнести то, что спецификация UML 1.3 не поддерживается в полной мере.

Среди других UML-редакторов выделяется продукт **Object Domain** компании Object Domain Inc. Этот редактор написан полностью на Java и выделяется из ряда других подобных систем скрупулезным следованием спецификации UML 1.3.

Кроме этого, Object Domain поддерживает внутренний объектный скриптовый язык (JPython), с помощью которого можно писать макросы, манипулирующие моделью и самим редактором, и интерфейсы для подключаемых модулей. Сейчас существует множество подключаемых модулей для Object Domain.

ArgoUML — это еще один UML-редактор, полностью написанный на Java. По возможностям он не слишком отличается от Object Domain кроме некоторой нестабильности работы и упомянут здесь только потому, что является open source проектом.

Существует коммерческий продукт, основанный на ArgoUML — Poseidon.

2. Интегрированные средства.

Одной из наиболее развитых интегрированных систем разработки, основанной на концепциях объектно-ориентированного подхода, является линейка AllFusion Modeling Suite. Это семейство продуктов компании Computer Associates включает в себя следующие компоненты.

• Process Modeler (BPwin) — один из ведущих инструментов визуального моделирования бизнес-процессов. Дает возможность наглядно представить любую деятельность или структуру в виде модели, что позволит оптимизировать работу организации, проверить ее на соответствие стандартам ISO9000, спроектировать оргструктуру, снизить издержки, исключить ненужные операции, повысить гибкость и эффективность. Являясь стан-

дартом de facto, BPwin поддерживает сразу три нотации моделирования: IDEF0 (федеральный стандарт США), IDEF3 и DFD.

• ERwin Data Modeler (ERwin) — продукт, являющийся de facto стандартом моделирования структур данных. ERwin позволяет проектировать, документировать и сопровождать базы данных, хранилища данных и витрины данных (data marts). Возможно создать наглядную модель базы данных, оптимизировать структуру базы данных и добиться её полного соответствия требованиям и задачам организации. Визуальное моделирование повышает качество создаваемой базы данных, продуктивность и скорость её разработки. Поддерживается прямое (создание базы данных на основе модели) и обратное (генерация модели по имеющейся базе данных) проектирование для 20 различных СУБД, настольных, реляционных и специализированных.

• Data Model Validator (ERwin Examiner) — инструмент для проверки структуры баз данных и создаваемых в ERwin моделей, позволяющий выявлять недочеты и ошибки проектирования. ERwin Examiner дополняет функциональность ERwin, автоматизируя трудоемкую задачу поиска и исправления ошибок, одновременно повышая квалификацию проектировщиков баз данных благодаря встроенной системе обучения.

• Model Manager (ранее ModelMart) — среда для совместной работы группы проектировщиков на AllFusion ERwin Data Modeler (ERwin) и/или AllFusion Process Modeler (BPwin) над одним проектом. Обеспечивает совместный доступ и редактирование моделей, повышая эффективность и скорость работы проектировщиков, является интегрирующим звеном для ERwin и BPwin. Защищает хранимые на сервере модели, позволяя задавать для сотрудников различный уровень доступа к ним, координировать весь ход работы над проектом.

• Component Modeler (Paradigm Plus) — CASE–средство для моделирования компонентов программного обеспечения и генерации объектного кода приложений на основе созданных моделей. Продукт можно использовать как при создании новых приложений, так и при изменении или объединении существующих. Благодаря интеграции с AllFusion Process Modeler (BPwin) есть возможность использования функциональной модели вместе с

объектной. Продукт поддерживает множество стандартных нотаций, таких как UML и Booch, интегрируется с технологиями COM/DCOM, CORBAPlus, BES VisiBroker и другими, продуктами CA, Microsoft, Rational Software. Paradigm Plus предоставляет возможности по эффективной визуализации, проектированию и сопровождению многоуровневых e–business приложений, обеспечивающих адаптацию к меняющимся требованиям. Возможности, предоставляемые продуктом, помогают пользователю выявить и решить следующие проблемы:

• полная поддержка Unified Modeling Language (UML);

• прямое и обратное проектирование. Paradigm Plus обеспечивает синхронизацию проектов приложений и их программных реализаций при любом числе изменений в коде и независимо от количества итераций проекта без применения маркеров кода и без потери данных;

• гибкость переноса информации. Комбинация XML, как источника данных, и спецификаторов шаблонов XSL предоставляет пользователю гибкость ввода и вывода информации в Paradigm Plus. Она также обеспечивает обмен информацией с другими продуктами. Paradigm Plus генерирует исчерпывающие настраиваемые отчеты, улучшающие взаимодействие между отдельными членами команды разработчиков. Поскольку Paradigm Plus обеспечивает полную поддержку XML и XSL шаблонов, то созданные отчеты могут быть легко переведены в различные форматы и размещены в Интернет;

• Paradigm Plus включает полностью документированный программный интерфейс приложения (API), который позволяет легко интегрировать его в существующую среду разработки;

• интеллектуальный редактор диаграмм. Paradigm Plus включает функцию "Интеллектуальные связи", которая упрощает создание крупномасштабных моделей за счет уменьшения числа действий, которые нужно совершать при рисовании связей;

• инструмент экспертного моделирования и встроенный помощник. Paradigm Plus повышает качество моделирования за счет поддержки UML в реальном времени и всплывающих подсказок, в которых отображается корректная сигнатура UML для каждого создания или изменения объекта.

Rational XDE представляет собой в некотором смысле дальнейшее развитие семейства продуктов Rational Suite. Это профессиональный инструмент, позволяющий разработчикам проектировать программные системы с нижнего, начального уровня, до готовой реализации. eXtended Development Environment — среда, интегрируемая в MS Visual Studio .NET, IBM WebSphere Studio Application Developer, позволяющая обычной среде разработки пользоваться технологиями проектирования Rational.

Rational XDE встраивается в среду разработки и позволяет осуществлять визуальное проектирование на основе диаграмм UML. По окончании процесса проектирования осуществляется генерация кода на выбранном языке программирования. Rational XDE осуществляет двустороннюю синхронизацию кода и модели, т.е. изменения, внесенные в модель отражаются в коде, а модифицированный код, воздействует на модель. Синхронизация осуществляется в двух режимах: ручном и автоматическом.

В состав Rational XDE входит модуль Rational Developer Accelerator (RDA), предоставляющий разрабатывать начальный код для программных проектов. RDA предоставляет набор расширяемых шаблонов, позволяющих быстро конструировать с нуля компоненты программной системы. Своими изобразительными возможностями и возможностями в генерации кода XDE дополняет Rational Rose.

Возможность поддержки множества моделей в Rational XDE Professional позволяет работать на различных уровнях абстракции. Разработчики могут создавать сценарии использования, проект архитектуры, модели реализации, которые будут одновременно отображаться в Rational XDE.

Rational XDE обеспечивает:

- двухстороннюю синхронизацию кода и модели с настраиваемым разрешением конфликтов как в ручном, так и в автоматическом режиме;
- четкое отображение элементов кода C-sharp (Java)в UML;
- более полное представление C--sharp (Java) приложения с помощью готовых моделей процессов Visual Studio .NET, что позволяет его верно понять;

• возможность одновременного отображения и кода, и модели;

• постоянную синхронизацию модели UML как части проекта C–sharp и Java.

Среди других интегрированных систем разработки, основанных на UML, стоит упомянуть TogetherJ и iUML.

Все они обладают своими особенностями, достоинствами и недостатками:

• TogetherJ — среда разработки, поддерживающая постоянное соответствие исходного кода системы и UML диаграммы. В некотором смысле можно сказать, что программирование системы идет полностью на UML. Это является как достоинство, так и недостатком системы, так как такой подход не дает возможности рисовать недоопределенные модели. Поддерживаемые языки программирования — Java, C++, C sharp, VisualBasic;

• В iUML. Среда ориентирована на real–time системы. Поддерживаются языки C++ и Java. Поддерживает язык описания действий, независимый от языка реализации, то есть одну и ту же модель можно сгенерировать как в Java так и в C++ код. Также поддерживается синхронизация кода и модели и эмуляция, или интерпретация модели, когда процесс отладки начинается еще до генерации кода прямо на модели.

3. Средства генерации подразделяются на средства генерации кода (на различных языках, в том числе и языках моделирования данных) и документации.

SoDA (Software Documentation Automation) — оригинальная разработка компании Rational, которая существенно упрощает и удешевляет процесс создания проектной документации и поддержания актуальности последней. SoDA особенно полезна при реализации крупных информационных проектов, в которых на составление документации и ее постоянную переработку часто тратится очень много времени и сил разработчиков. По задаваемым пользователем шаблонам SoDA "компилирует" документацию, собирая в один документ текстовые и графические данные из различных источников, в частности из моделей, созданных в Rational Rose, после чего пользователь может отредактировать

этот документ с помощью Microsoft Word или Adobe FrameMaker+SGML.

Основные возможности, обеспечиваемые SoDA:

• автоматическое извлечение информации из файлов, созданных различными инструментальными средствами. SoDA "понимает", структуру информации, хранимой теми системами, с которыми она интегрирована — это, в частности, FrameMaker и Microsoft Word;

• сохранение при "перекомпиляции" текста и графики, введенных пользователем вручную в текстовом процессоре. Если пользователь, допустим, в Microsoft Word, добавил какие–нибудь комментарии или иллюстрации в сгенерированный с помощью SoDA документ, то генерируя этот документ снова, SoDA их не "испортит";

• настройка шаблонов, по которым генерируется документация. С помощью WYSIWYG–редактора можно создавать шаблоны, соответствующие всевозможным внешним стандартам (таким, как ISO 9000, IEEE, MIL–STD–498 и DOD–STD–2167A) или внутренним стандартам компании. Синхронизация с источниками и проверка актуальности документации. Связи между отдельными частями документации и исходными файлами запоминаются. Поэтому, во–первых, SoDA может отслеживать изменения, происходящие с источниками, на основе которых была в последний раз "скомпилирована" документация, и, во–вторых, пользователь может, находясь в любой секции документа, быстро получить доступ к источникам, информация из которых используется в этой секции;

• частичная "перекомпиляция" больших документов. Проектная документация к масштабным программным системам может достигать гигантских объемов. Поэтому в SoDA предусмотрена возможность "перекомпилировать" только те части документации, которые действительно утратили актуальность;

• сбор информации из многочисленных и разнородных источников и документирование всех этапов работы над проектом. SoDA интегрирована с инструментальными средствами, используемыми на всех стадиях разработки программного обеспечения — от определения требований к системе до тестирования;

• проверка соблюдения требований, предъявляемых к разрабатываемой системе. SoDA позволяет сформировать таблицы, из которых можно понять, насколько соответствует требованиям, определенным на начальных этапах проектирования, то, что было сделано впоследствии.

Семейство UML–продуктов компании Novosoft Inc: Zebra, FL, ZServlets. Все эти продукты представляют собой генераторы, основанные на библиотеке NGEN и предназначенные для создания Java приложений.

Входная модель, по которой генерируется код, может быть представлена в одном из двух наиболее популярных форматов представления моделей: XMI (из стандарте UML 1.3) и Petal (формат Rational).

Каждый из этих продуктов покрывает определенную часть функциональности приложения, и может быть использован как отдельно, так и совместно с другими.

FL представляет собой генератор Java классов, входящих в слой доступа к базе данных (persistence layer), по UML модели. FL позволяет практически полностью скрыть уровень доступа к конкретной базе данных за объектным интерфейсом, что существенно облегчает в дальнейшем миграцию на другую базу данных и упрощает написание кода, работающего с данными. Основная идея FL это отображение модели данных приложения, выраженной в виде набора UML классов и ассоциаций, на реляционную базу данных; причем доступ к этой базе данных инкапсулирован в java объектах, сгенерированных по UML модели. Это значит, что работа с базой данных выглядит для программиста практически так же, как работа с обычными java объектами. Обычно модель, из которой в дальнейшем будет сгенерирован код и классы для доступа к базе данных часто очень похожа на модель бизнес–объектов.

Важной чертой FL является возможность использования OQL — Object Query Language. OQL — язык запросов, позволяющий формулировать запросы в терминах объектов, их атрибутов и ассоциаций. В дальнейшем FL генерирует SQL–запросы к базе данных по OQL–запросам, что позволяет разработчику полностью абстрагироваться от уровня конкретной базы данных.

Основные черты продукта.

1. Генерация структуры базы данных и Java объектов доступа к ней по UML–модели данных. При генерации каждому классу модели ставится в соответствие таблица в базе данных.

2. Поддержка всех примитивных типов данных Java, типов String, Date, FLBlob и FLClob.

3. Поддержка ассоциаций и наследования.

4. Поддержка различных баз данных: DB2, Interbase, MSAccess, MSSql, Oracle, Informix.

5. Поддержка OQL.

6. Поддержка обратных ассоциаций (поиск объектов, ссылающихся на данный или данные).

7. Наличие инструментов проверки целостности базы данных.

8. Пулинг соединений с базой данных.

9. Возможность использования классов, сгенерированных FL, в J2EE контейнерах, в том числе и в кластерном окружении, что позволяет увеличить производительность и стабильность.

Zebra, в отличие от FL, позволяет генерировать Java классы, реализующие поведение приложения, в некотором смысле FL и Zebra эффективно дополняют друг друга. Основными элементами модели, из которой Zebra генерирует код, являются классы и машины состояний, задающие поведение объектов классов. Кроме развитого и хорошо определенного расширения языка UML собственно генератора кода, Zebra также предоставляет компоненты времени исполнения, ускоряющие разработку приложения.

Важными чертами продукта являются:

1. Полное соответствие стандарту UML 1.3.

2. Возможность прямой и обратной синхронизации модели и кода.

3. Четкое выделение слоев приложения.

4. Наличие компонент времени исполнения, предоставляющих функциональность и модули, часто встречающиеся в реальных приложениях, такие как подсистема безопасности, управление сессиями пользователя, управление потоками и т.п.

5. Наличие презентаций — заменяемых модулей, реализующих различные типы пользовательского интерфейса системы: swing презентация (GUI), JSP презентация (web интерфейс), XML презентация (SOAP интерфейс) и других.

ZServlets представляет собой специализированное расширение Zebra, предназначенное для генерации сервлетов по UML моделям. Java сервлет генерируется по классу, для которого определяется машина состояний, реагирующая на события HTTP протокола — GET, POST, PUT и прочие.

4. Метаинструменты.

NS UML (Novosoft UML API) является базовой библиотекой для многих продуктов, в том числе описанного выше семейства продуктов NGEN, и самого инструмента NGEN, UML редакторов ArgoUML и Poseidon, Object Domain и многих других. NS UML представляет собой Java модуль, обеспечивающий работу с UML моделями в Java приложениях. Основными чертами продукта являются:

- NSUML предоставляет быстрый и удобный доступ к модели, обеспечивает эффективную реализацию внутреннего представления модели, стандартные интерфейсы для реализации других способов представления;
- API позволяет производить с UML моделями все необходимые операции: генерация, сериализация, доступ к элементам модели, их модификация, добавление и удаление. Также NSUML автоматически поддерживает целостность модели;
- Novosoft UML API удовлетворяет всем спецификациям OMG/UML, реализованы все элементы UML версии 1.3: пакеты, типы данных, классы, их методы и ассоциации и т.д. Кроме того, в API определено множество полезных утилитных методов не специфицированных OMG;
- NSUML содержит так называемый Reflective API. Вместе со стандартными методами доступа, это дает разработчику множество дополнительных возможностей доступа к модели и ее элементам. Иногда это существенно упрощает работу с UML моделями.
- в NSUML есть поддержка нотификации о событиях модели (в основном это события, связанные с изменениями модели);

• поддержка undo/redo. Программист может ставить контрольные точки (или точки возврата), после чего возможен возврат к состоянию модели или ее части на момент установки точки;

• поддержка стандарта XMI. Модель может быть прочитана из XMI файла, и сохранена в виде файла XMI.

NSUML широко используется для разработки генераторов кода по UML моделям для различных языков, таких как Java, C++, SQL и т.д. Так же продукт может быть весьма полезен разработчикам различных инструментов программирования в рамках объектно–ориентированной парадигмы.

Частью NSUML, и в то же время независимой библиотекой, является Novosoft Petal File API. Novosoft Petal File API представляет собой Java модуль, позволяющий читать файлы, содержащие описание UML модели в формате Petal. Этот формат используется одним из наиболее популярных семейством инструментов UML разработки, созданным компанией Rational, в частности программой Rational Rose.

Novosoft Petal File API читает файл в формате petal, разбирает его и преобразует во внутреннее представление NSUML.

Наличие Novosoft Petal File API позволяет использовать все основанные на NSUML программные продукты вместе с наиболее известными UML инструментами разработки фирмы Rational.

NGEN, или Novosoft Generator Framework, представляет собой Java модуль, содержащий простой генератор Java кода по UML моделям и, что самое главное, позволяющий разработчику строить собственные генераторы.

NGEN полностью основан на NSUML. В качестве входных данных NGEN получает NSUML модель, после чего, в зависимости от конфигурации обрабатывает ее сам, генерируя Java классы, либо отдает модель или ее части подключенным модулям–генераторам.

Краткое описание основных черт продукта.

• Продукт основан на NSUML и использует в качестве входных данных NSUML–модель.

• Гибкость конфигурирования форматов входных файлов. NGEN позволяет подключать и использовать различные модули

чтения входных файлов. Так, в зависимости от необходимости, модель может передаваться генератору в формате XMI и разбираться стандартным модулем чтения NSUML, или же модель может передаваться в формате Petal (формат Rational Rose) и разбираться модулем чтения Petal файлов (Novosoft Petal File API). И в том и в другом случае модель будет переведена во внутреннее представление NSUML.

• Гибкость конфигурирования модулей–обработчиков модели. NGEN позволяет подключать и использовать различные модули обработки модели (в частности модули генерации кода). В частности, FL и Zebra являются такими модулями, и могут быть использованы в NGEN как вместе, так и по отдельности.

• Наличие стандартного модуля–генератора, позволяющего генерировать шаблоны Java классов по UML–модели.

• Наличие минимального графического интерфейса.

Заключение

В статье кратко рассмотрены современные технологии и инструменты построения больших программных систем на основе объектно-ориентированного подхода и языка UML, описан процесс разработки rational unified process.

Дана верхнеуровневая классификация инструментов разработки, основанных на UML и приведены в качестве примеров наиболее популярные и известные представители каждого класса.

Кроме того, в рамках данной классификации кратко описано семейство подобных продуктов фирмы Novosoft Inc.

Литература

1. BOOCH G. Object–Oriented Analysis and Design with Applications, 2nd edition. The Benamin Cummings – Redwood City. — 1994.

2. JACOBSON I. The Unified Software Development Process. – Addison–Wesley, 1999.

3. BOOCH G. and others. The Unified Modeling Language User Guide/ G.Booch, J.Rumbaugh, I.Jacobson, J.Rumbaugh. – Addison–Wesley. — 1998.

4. RUMBAUGH J. Object–Oriented Modeling and Design.– Prentice Hall. — 1991.

5. RUMBAUGH J. OMT Insights. – Cambridge: Cambridge University Press, 1996.

6. JACOBSON I. Object–Oriented Software Engineering: A Use Case Driven Approach.– Addison–Wesley: ACM Press,1993.

7. JACOBSON I. The Object Advantage: Business Process Reengineering With Object Technology.– Addison–Wesley: ACM Press, 1995.

8. JACOBSON I. and others. Software Reuse: Architecture Process and Organization for Business Success/ I.Jacobson, M.Griss, P.Jonsson. – Addison–Wesley: ACM Press, 1997.

9. JACOBSON I., BYLUND S. The Road to the Unified Software Development Process.– Cambridge: Cambridge University Press, 2000.

10. Meilir Page-Jones. Fundamentals of Object–Oriented Design in Uml (Addison Wesley Object Technology Series), ISBN 020169946X, Addison–Wesley. — 1999.

11. ИВАНОВА Г.С. и др. Объектно–ориентированное программирование: Учебник (под ред. Ивановой Г.С.) Информатика в техническом университете / Иванова Г.С., Ничушкина Т.Н., Пугачев Е.К.– М: МГТУ им. Н.Э.Баумана, 2001.– 320 с.

12. ФАУЛЕР М., СКОТТ К. UML: Основы: Краткое руководство по унифицированному языку моделирования: Пер. с англ. Изд. 2–е. – С–Петербург: Символ–Плюс,2002.– 192 с.

13. КВАТРАНИ Т. Rational Rose 2000 и UML: Визуальное моделирование: Пер. с англ. Объектно–ориентированные технологии в программировании. – М: ДМК Пресс. — 2001.– 176 с.

14. БУЧ Г. и др. UML: Специальный справочник: Пер. с англ. / Рамбо Дж., Якобсон А., Буч Г.– С–Петербург: Питер, 2002.– 656 с.

15. КРАТЧЕН Ф. Введение в Rational Unified Process: Пер. с англ. Изд. 2-е. Объектные технологии.- Москва–С.-Петербург: Вильямс,2002.- 240 с.

Поступила в редакцию
5 июня 2002 года