Общие принципы построения модели в Rational Rose

Цель данной статьи - описать общие принципы построения диаграмм Rational Rose (далее RR), их назначение и связь диаграмм между собой. Предложенный подход достаточно прост в понимании и позволяет представить для чего и как строить диаграммы RR.

Предположим, что нам поручено провести анализ бизнес-процессов у заказчика и на основе данного анализа построить модель информационной системы (ИС).

Мы должны провести анализ документооборота, состав документов, роли ответственных лиц (Actors) и т.д.

Задача это не простая и требует значительных аналитических усилий и опыта.

Результатом этой работы должен быть список ролей в компании заказчика, четкое понимание процесса и список объектов (сущностей), участвующих в этом процессе.

Все это и должно найти отражение в диаграммах RR. Кроме того, мы вместе с заказчиком должны составить список требований к ИС.

Общий подход таков:

используем ***Use case diagram*** для отображения списка операций, которые должна выполнять наша система; иначе говоря, это требования к системе.

Каждый *Use case* – это некоторый процесс (последовательность действий), поэтому мы должны использовать

Sequence diagram для его детализации. На этой диаграмме мы отображаем объекты из предметной области (объекты, участвующие в бизнес-процессе); таким образом, мы получаем экземпляры некоторых классов и их взаимодействие. *Sequence diagram* отображает сам процесс, статическая картина взаимодействия объектов отображается с помощью ***Class diagram.***

Переходим к ***Class diagram***, на которой изображаются классы нашей ИС. Далее классы объединяются в компоненты,

которые отображаются на ***Component diagram***, где показывается зависимость компонентов между собой.

На ***Deployment diagram*** отображается размещение этих компонентов по компьютерам (узлам сети) для проектируемой ИС.

Вот и все, т.е., мы движемся от ***Use case diagram --> Sequence diagram --> Class diagram --> Component diagram --> Deployment diagram.*** Этот набор диаграмм должен присутствовать всегда в моделях RR, остальные типы диаграмм нам пригодятся для большей детализации нашей модели.

Но давайте более подробно. Итак, мы имеет

1. ***список требований*** к ИС составленный с помощью Requisite(R Pro) или Doors

составить такой список очень просто: открываете проект в Requisite (R Pro (Project --> Open) и выбираете в меню команду Document --> New.

Открывается документ MS Word, где Вы в табличном виде (думаю так лучше) формируете список требований (из контекстного меню выбираете ”Create Requirement…” для каждого требования).

2. Начнем создавать диаграмму прецедентов - Use case diagram.

На *Use case diagram* отображаем взаимодействие между ролями (актерами) и прецедентами (т.е. это случаи использования ИС).

Например, фраза “Заказчик формирует заказ на доставку товара” приводит нас к пониманию следующего: Актер – “Заказчик”, прецедент – “Сформировать заказ на доставку” (прецедент обычно начинается с глагола), требование к системе - “ИС должна поддерживать операцию формирования заказа на доставку” (список требований ранее сформирован). Теперь необходимо изобразить всех актеров и те действия, которые они могут выполнять (т.е. прецеденты) на одной или нескольких ***Use case diagrams,*** **а затем связать каждый прецедент с требованием,**

ранее сформированном в Requisite(R Pro (используйте контекстное меню на ***Use case:*** **Requirement Properties --> Associate…).**

3. Прецедент – это некоторый процесс, в котором обычно участвуют несколько объектов, т.е. “Сформировать заказ на доставку” предполагает некоторую *последовательность операций:*

- открыть бланк заказа --> заполнить реквизиты заказчика --> заполнить тип и количество товара --> отправить заказ на исполнение (или отменить заказ или еще что-то).

Вот этот процесс мы и должны отобразить на ***Sequence diagram.*** На этих диаграммах слева всегда изображают одного Actor, далее стоят последовательно объекты (или ассоциированные с ними классы), а стрелочками отображается передача сообщения (или вызов метода класса).

Например, передача сообщение “Открыть бланк заказа” отображается в виде стрелки от актера “ЗАКАЗЧИК” к объекту “ЗАКАЗ” с соответствующей записью на диаграмме. Сразу заметим, что при переходе на *Class diagram* мы создадим класс COrder с методом NewOrder() для данного сообщения.

В общем, проведенный нами анализ бизнес-процессов у заказчика должен отобразиться в диаграммных последовательностях действий - ***Sequence diagrams.***

Мы может создать эту диаграмму для каждого прецедента (Open

Specifications --> Diagrams --> Insert Sequence diagram) или общую на группу прецедентов (New --> Sequence diagram).

4. ***Построить Collaboration diagrams*** (диаграммы взаимодействий) достаточно просто по *Sequence diagrams* (они также детализируют процесс взаимодействия, но не представляют его во времени): Browse --> Create Collaboration diagram (F5). Имея набор Sequence diagrams, мы можем переходить к проектированию классов и построению диаграмм классов, которые представляют статическую картину взаимодействия объектов.

5. Мы должны для каждого объекта создать класс, в котором будет реализовано поведение этого объекта через методы класса.

Каждый объект в *Sequence diagram* должен быть ассоциирован с классом: откройте спецификацию объекта и укажите класс, затем для каждого сообщения на *Sequence diagram* замените его имя на имя метода из класса.

Для проверки модели выполните Tolls --> Check Model.

Построение диаграмм классов (*Class diagrams*) является самым важным и трудоемким этапом в создании модели. Понятно, что поведение объекта не возможно встроить в один класс, поэтому необходимо создавать диаграммы классов, устанавливая связи между классами. Каждый класс имеет набор методов (Operations) и переменных (Attributes). В UML принято несколько типов связей (отношений) между классами, причем их название и интерпретация отличаются от принятых в ООП, но кратко их можно описать так:

Association – семантическая связь между классами, показывает передачу сообщений между классами, при генерации кода в определение класса добавляется переменная класса, на который направлена ассоциация;

Dependency – показывает зависимость одного класса от определений в другом классе, например, когда один класс используется как параметр в описании методов другого класса, при генерации кода не вносит изменений в описание класса;

Aggregation – связь между целым и его частями, при генерации кода в определение класса добавляется переменная другого класса, являющейся частью;

Generalization – связь наследования между классами, соответствует понятию наследования классов в ООП;

По диаграмме классов мы можем провести генерацию проекта (набора, например, .h и .cpp-файлов), при этом важно настроить свойства класса, которые влияют на кодогенерацию (вкладки “C++”, “MSVC” и другие). Понятно, что имплементации методов никакой не будет, только их определения.

6. Имея набор классов, мы может перейти к формированию компонентов ИС – физические модули ИС (DLL, EXE и другие модули). Зависимости между компонентов показываются на *Component diagram*.

Каждый компонент ассоциирован с одним или несколькими классами, которые и определяют содержимое компонента.

Далее откройте спецификацию компонента и на вкладке “Realizes” назначьте классы для данного компонента (из контекстного меню ”Assign”), а также на вкладке “General” укажите язык программирования (теперь спецификация классов дополнится новыми вкладками, влияющих на кодогенерацию для данного языка программирования).

7. Deployment diagram (диаграмма развертывания) построить достаточно просто, т.к. она не содержат привязки к компонентам в модели RR, т.е. не обязательно строить эту диаграмму на последнем этапе проектирования. На этой диаграмме изображаются компьютеры (Processor) и связи между ними.

Чтобы отобразить задачи, которые выполняются на этих компьютерах (узлах сети) откройте спецификацию и на вкладке “Detail” для поля “Processes” введите имена задач, которые могут соответствовать именам компонентов проектируемой ИС.

Процесс построение модели завершен. Выполним проверку модели с помощью команды Tolls --> Check Model и посмотрим результаты в Log окне. Если ошибок нет, можно переходить к генерации проекта и ипмплементации ИС.

Процесс построения модели носит итерационный характер. Невозможно построить модель сразу, часто необходимо возвращаться на другие уровни представления информации, вносить изменения, детализировать и снова возвращаться на уровень имплементации (диаграммы классов уровня генерации кода).

К тому же ИС подвержена постоянному изменению (причины вам известны), поэтому и **код ИС и ее модель должны меняться согласованно и соответствовать друг другу.**