СБОРНИК **НАУЧНЫХ ТРУДОВ НГТУ. – 2010. – № 3(61). –** 77–84

УДК 519.6

ИНТЕГРАЦИЯ СТРУКТУРНЫХ И ДИНАМИЧЕСКИХ UML-МОДЕЛЕЙ*

И.В. ЗИМАЕВ

На примере проектирования диспетчера процессов рассматривается возможный способ сопряжения диаграммы классов и диаграммы деятельности, применяемый для последующей автоматической трансляции полученного синтеза в верифицирующую цветную сеть Петри.

Ключевые слова: диаграммы UML, моделирование, сети Петри, программная инженерия.

ВВЕДЕНИЕ

Одним из наиболее эффективных подходов при разработке программного обеспечения является объектно-ориентированное проектирование (ООП). Эта технология прошла многолетний путь эволюции, найдя свое массовое воплощение во множестве языков программирования, всевозможных интегрированных средах разработки, пакетах имитационного моделирования и пр. Ключевое понятие данного подхода – объект, им обозначают программный аналог некоторого предмета реального мира или предметной области. Основной характеристикой объекта являются данные. В широко используемом унифицированном языке моделирования UML для представления структуры данных предусмотрен ряд диаграмм [1], обобщенно называемых статическими, или структурными (например, диаграмма классов). Назначение этих диаграмм – показать архитектуру программы на различных масштабах в виде взаимосвязанных блоков информации. Для описания алгоритмов работы программы предусмотрены динамические диаграммы. Как правило, явная связь между структурным и динамическим представлением модели отсутствует. Объединение этих представлений предлагается как одна из составляющих метода автоматической верификации архитектуры программы с помощью имитационного моделирования сетями Петри.

* Статья получена 4 октября 2010 г.

Работа выполнена при финансовой поддержке Министерства образования и науки РФ, конкурс НК-81П, Гос. контракт № П694 от 12.08.2009.

1. ПОРЯДОК ПРОЕКТИРОВАНИЯ

Началом процесса объектно-ориентированного проектирования программной системы является создание структурной диаграммы. После достижения некоторой степени проработанности структуры разработчик переключает свое внимание на динамический аспект архитектуры и описывает алгоритм работы системы. Язык UML предоставляет широкий набор динамических диаграмм, также часто называемых поведенческими (например, диаграмма деятельности). Приведенный порядок проектирования, в котором сначала разрабатывается структура, затем динамика, весьма удобен, а главное, логичен. Под логичностью здесь подразумевается корреляция между объектно-ориентированной парадигмой и познавательной деятельностью человеческого разума. В процессе познания любого явления или объекта можно условно выделить четыре основные стадии (рис.1), в соответствии с которыми целесообразно организовать и обратный процесс – моделирование новой системы.

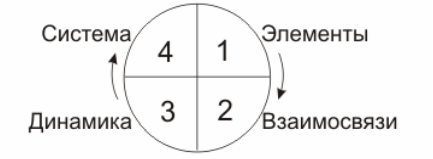


Рис. 1. Этапы процесса моделирования

Системный подход к моделированию программной системы начинается с описания ее структуры, множества составляющих ее элементов. Далее между элементами устанавливаются взаимосвязи. Последующие две стадии относятся к динамике: на третьей стадии описывается процесс взаимодействия взаимосвязанных элементов, и завершающий этап – это синтез знаний о структуре, связях и динамике элементов в одно целое – систему. Таким образом, любая единица программного обеспечения (функция, класс, модуль, программа) может рассматриваться как система со всеми присущими системам свойствами (такими, как целостность, иерархичность, эмерджентность и пр.).

Структурная модель системы отражает набор ее элементов и их взаимосвязи, динамическая модель – отражает процессы взаимодействия элементов между собой и с внешней средой. Важно отметить, что динамическая модель отчасти включает в себя структурную, поскольку каждое действие программной системы связанно с какими-либо данными – элементами. Для полного описания структуры какой-либо системы не требуется описание ее поведения.

2. ВЗАИМОСВЯЗЬ ДИАГРАММ

Хотя структурные и динамические диаграммы описывают одну и ту же систему, только с разных ракурсов, четкая связь между этими диаграммами в языке UML отсутствует. Сам по себе этот факт заслуживает внимания. Структуру любой системы можно формально представить в виде множества ее элементов и множества связей между ними (при этом каждый элемент сам также может быть рассмотрен как система элементов). Поведение системы хорошо описывается с использованием теории машины состояний и теории графов. Например, диаграмма деятельности фактически является орграфом, состоящим из множества действий (вершины) и их связей (дуги). Очевидно, что взаимосвязь структуры и динамики также может быть до некоторой степени формализована, в частности, в виде множества кортежей, каждый из которых показывает, какие элементы структуры в какие действия вовлекаются. Качество или виды связи между действиями и элементами структуры могут быть различны, в частности, это отношение ассоциации и отношение композиции. Под ассоциацией, аналогично этому понятию в языке UML, подразумевается участие данного структурного элемента в действии (т.е. без элемента действие не состоится). Композиция обозначает управление действия временем жизни элемента (т.е. элемент создается или разрушается в процессе действия).

Связь динамики со структурой предоставляет дополнительную информацию о работе системы, отражает ее внутренние процессы, что крайне ценно для имитационного моделирования и верификации. Поэтому одним из перспективных применений подобного синтеза диаграмм является решение задачи автоматической трансляции UML-диаграмм в сети Петри.

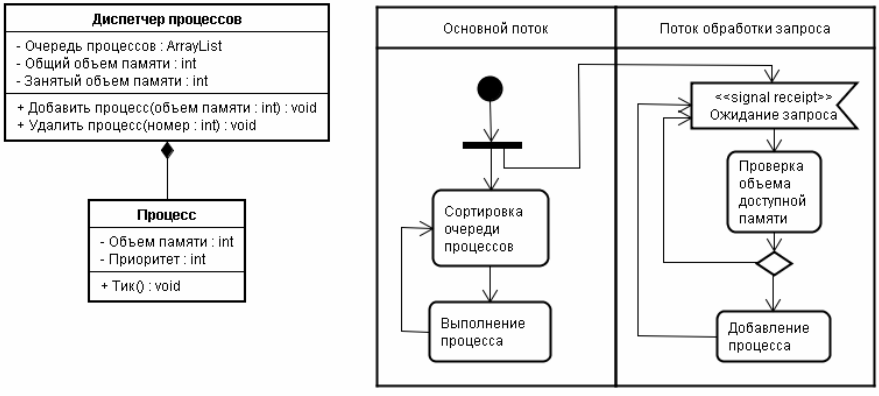
3. ТРАНСЛЯЦИЯ ДИАГРАММ В СЕТЬ ПЕТРИ

При помощи математического аппарата сетей Петри возможна [2] верификация архитектуры программной модели. Для этого исходная поведенческая диаграмма (как правило, это диаграмма состояний или диаграмма деятельности) согласно набору правил [2] преобразуется в сеть Петри, позволяющую осуществить имитационное моделирование. Если при работе сети возникли ошибки (например, тупиковые ветви), значит, при проектировании архитектуры тоже были допущены ошибки, и результаты моделирования сети Петри позволяют их локализовать [3]. В условиях все возрастающей сложности разрабатываемого программного обеспечения подобный подход призван значительно повысить его качество. Однако на сегодняшний день задача автоматической трансляции UML-диаграмм в сети Петри полностью не решена. Обеспечить трассовую эквивалентность исходной диаграммы и сети Петри на

уровне автоматизации возможно [4], однако немалая часть информации о поведенческой модели системы не представима структурой графа деятельности, и при трансляции она теряется [5]. Если между динамической и структурной диаграммами определены взаимосвязи, то при автоматической трансляции первой в сеть Петри эти связи способны передать больший объем информации о поведении системы.

4. ПРОЕКТИРОВАНИЕ ДИСПЕТЧЕРА ПРОЦЕССОВ

Основная задача диспетчера процессов – распределение процессорного времени между запущенными приложениями (согласно какому-либо алгоритму многозадачности). Соответственно ключевыми объектами в данном проекте являются процесс и диспетчер. Диаграмма классов может выглядеть так, как изображена на рис. 2,*а*.



а *б*

Рис. 2. Диаграмма классов (*а*) и деятельности (*б*)

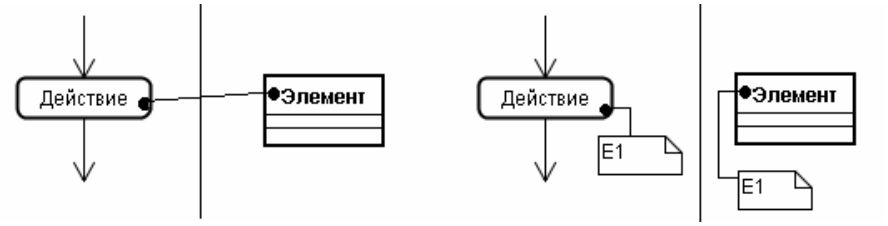
Класс процесса содержит такие данные, как приоритет и объем занимаемой им памяти. Класс диспетчера содержит список запущенных процессов, общий и занятый объем памяти. Поскольку в классе диспетчера будут создаваться и храниться объекты класса процесса, на диаграмме между этими классами показана композитная связь.

Простейшая блок-схема работы диспетчера представлена на рис. 2,*б*. У диспетчера предусмотрено два внутренних потока работы (разделение происходит сразу после старта при помощи элемента FORK). В первом потоке осуществляется цикл обработки запущенных процессов из двух стадий: сор-

тировка списка процессов по приоритету и выполнение процесса (выделение процессорного времени). Во втором потоке происходит проверка возможности выделения запрошенного объема памяти и добавление процесса в общий список.

5. СИНТЕЗ ДИАГРАММ

Теперь возможно связать полученные модели статики и динамики посредством объединенной диаграммы. На синтезированной схеме одновременно изображаются диаграмма классов и диаграмма деятельности. Связь определенного действия (элемента динамической диаграммы) с элементом структуры можно изобразить соединительной линией (рис. 3,*а*).



а *б*

Рис. 3. Варианты представления связи диаграмм

Второй способ (рис. 3,*б*) заключается в том, что ряд элементов структурной диаграммы снабжается маркерами, и эти же маркеры указываются для элементов диаграммы деятельности. Этот способ обозначения выбран в целях сохранения читаемости диаграмм, ведь если показывать связи между диаграммами в виде линий, они могут затруднить восприятия схемы. Качественно разные типы связи между диаграммами также обладают визуальными отличиями на схеме. Пунктирная линия означает связь ассоциации (обращение на чтение или запись) с соответствующей структурой данных (элементом), сплошная линия – означает создание объекта в результате действия.

Интегрированная диаграмма для рассматриваемой задачи диспетчера процессов принимает вид, изображенный на рис. 4.

При трансляции полученного синтеза диаграмм в сеть Петри основным источником информации является диаграмма деятельности. Связи со структурой встраиваются в сеть Петри дополнительными элементами, моделирующими процессы работы с данными.

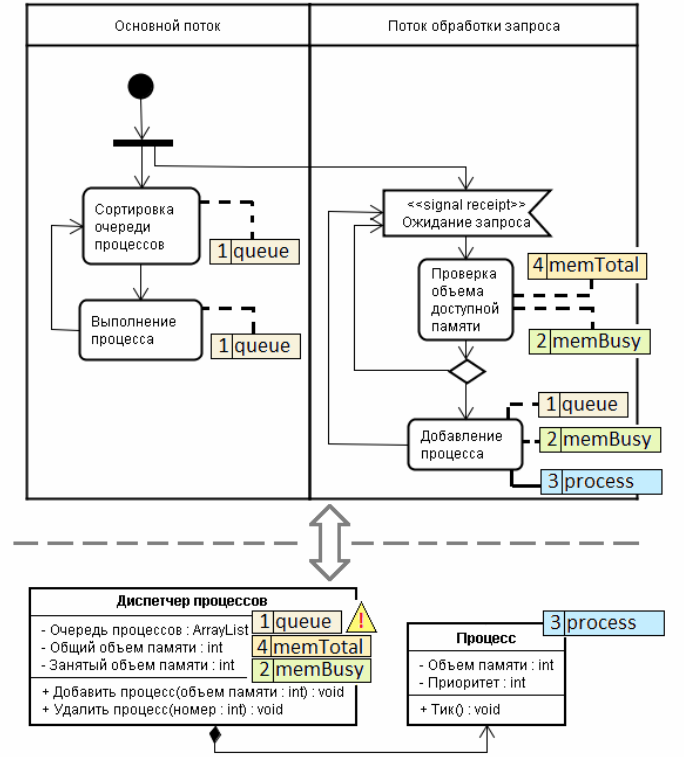


Рис. 4. Синтез структурной и динамической диаграмм

6. СЕТЬ ПЕТРИ

Результат трансляции интегральной диаграммы в сеть Петри можно увидеть на рис. 5.

Трансляция осуществляется по известным [2, 4] правилам, которые дополнены еще одним – правилом трансляции связи действия с элементом структуры данных: *если действие A через отношение ассоциации связано с элементом данных E, в сети Петри действие A представляется переходом, а элемент E – позицией (местом). Для позиции задается свой цвет (тип данных, указанный на диаграмме) и один начальный маркер, переход A дополняется переходом перед ним и после него с промежуточными связующими позициями. Первый дополнительный переход изымает маркер из позиции E, второй переход его возвращает.*

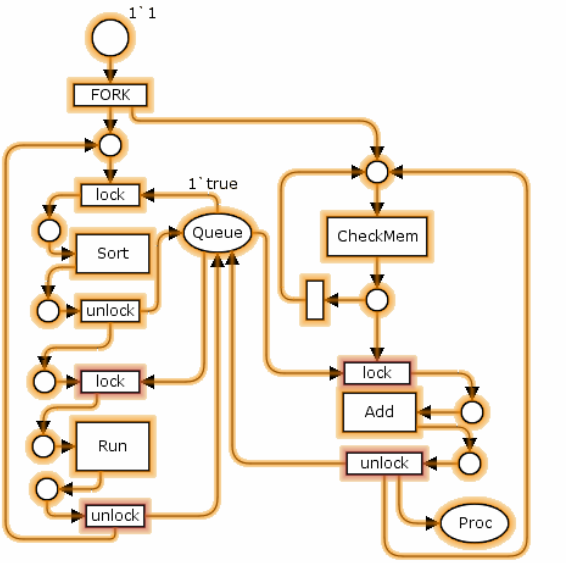


Рис. 5. Сеть Петри для интегральной диаграммы

Работа сети начинается с перехода стартового маркера в две позиции, что имитирует дальнейшую работу двух потоков. В основном потоке последовательно выполняется две операции: сортировка и выполнение процессов, каждое из этих действий сопровождается обращением к списку процессов, что имитируется изъятием маркера из позиции Queue. Если началась сортировка списка (сработал переход Lock перед действием Sort), позиция Queue становится пустой, следовательно, никакое действие с участием переменной Queue в программе невозможно. Аналогичным образом блокировка на работу с Queue устанавливается операций Run и Add, причем вторая находится в дополнительном потоке. Очевидно, что конфликта одновременного доступа к Queue между действиями одного потока (Sort и Run) возникнуть не может из-за их разнесения во времени, вытекающего из их строгой очередности выполнения. Однако ситуация, когда со списком процессов работает функция Sort и в этот же момент в параллельном потоке происходит обращение к списку функцией Add, вполне возможна, и имитационное моделирование сети Петри это демонстрирует. С точки зрения функционирования сети Петри данная архитектура не имеет тупиковых веток, поэтому, для выявления ошибки параллельного доступа необходимы дополнительные ограничения на работу сети.

ЗАКЛЮЧЕНИЕ

Системный подход к описанию структурных и поведенческих диаграмм призван формализовать модель архитектуры программного обеспечения, и, будучи дополненным указанием связей между диаграммами, позволяет увеличить качество автоматической трансляции UML-диаграмм в сеть Петри. При имитационном моделировании полученной сети становится возможным выявление ряда ошибок, особенно критических для модели многопоточных вычислений. В частности, если несколько параллельных действий в программе связаны с обращением к одним и тем же структурам данных, возможен конфликт одновременного доступа. На диаграмме эта ошибка проектирования может остаться незамеченной, но при работе верифицирующей сети Петри она будет выявлена.

[1] *Ларман К.* Применение UML 2.0 и шаблонов проектирования. – 3-е издание / Ларман, Крэг; пер. с англ. – М.: ООО «И.Д. Вильямс», 2007. – 736 с.

[2] *Коротиков С.В.* Применение сетей Петри в разработке программного обеспечения центров дистанционного контроля и управления: дис. …канд. техн. наук. – Новосибирск: НГТУ, 2007.

[3] *Воевода А.А., Зимаев И.В.* Моделирование системы многоканальной визуализации с использованием аппарата сетей Петри // Сб. науч. тр. НГТУ. – № 3(53). – 2008. – С. 45–50.

[4] *Воевода А.А., Зимаев И.В.* Об особенностях преобразования UML-диаграмм деятельности в сети Петри // Сб. науч. тр. НГТУ. – № 2(56). – 2009. – С. 79–89.

[5] *Zimaev I.V.* Automatic reflection of activity diagrams in colored Petri nets // International Forum on Strategic Technology (IFOST), 2010, Ulsan, South Korea.

Зимаев Игорь Валентинович, аспирант кафедры автоматики НГТУ. Тема научной работы – использование UML диаграмм и аппарата сетей Петри как формальных методик анализа архитектуры программного обеспечения. Имеет 6 публикаций. E-mail: topspace@mail.ru

Zimaev I.V.
Integration of structural and dynamic UML-models

Article considers an approach of integration class diagrams and activity diagrams. Obtained synthesis translates automatically in colored Petri net for verification.

Key words: UML diagrams, modeling, Petri nets, software engineering.