ПРИМЕНЕНИЕ СУБЪЕКТОВ В ОБЪЕКТНО-ОРИЕНТИРОВАННОМ ПРОЕКТИРОВАНИИ И ПРОГРАММИРОВАНИИ

А.В.Тарасевич[1], Е.И. Велесько[2]

[1] - студент 2 курса, факультета прикладной математики и информатики, 3 группы Белорусского государственного университета

[2] - научный руководитель, кан.экон.наук, профессор кафедры экономики и управления, Белорусского государственного экономического университета, Минск, 220672, Партизанский пр., 26, тел.(8017) 249 81 91, e-mail: velesko@bseu.minsk.by

Аннотация. В работе рассматривается подход к разработке программ, позволяющий несколько упростить проектирование, уменьшить количество допускаемых ошибок и упростить поиск ошибок за счет четкого выражения отношений между элементами программной системы. В соответствии с этим подходом с каждым значительным элементом создаваемой программы связывается набор функций, которые он выполняет, а также права и обязанности по отношению к элементам, с которыми он взаимодействует.

Ключевые слова: Субъект, контракт, объектно-ориентированное проектирование и программирование (ООП), объект, ресурс, ошибки, ответственность, права, обязанность, взаимодействие.

Понятие субъекта

Субъект – это «одушевленный» элемент программной системы, несущий ответственность за осуществление некоторой функции (задачи), обладающий правами и обязанностями в отношении других субъектов. Перед субъектом стоит две цели:

1. Обеспечить соответствующую ему функциональность
2. Не вступать в конфликт с другими субъектами

Субъекты вводятся на основании задач, которые они должны решать. При этом понятие «задача» используется в широком смысле слова и может подразумевать как вычисление арифметической операции, так и управление такой сложной системой, как промышленное предприятие. При проектировании субъекты позволяют полностью абстрагироваться от деталей реализации и просто считать субъектом нечто, что решает определенную проблему. Субъектом можно считать и переменную, и класс, и библиотеку, и целый программный комплекс. Субъект не является новой сущностью, это лишь определенное понимание других сущностей.

В чем принципиальное отличие между объектом в понимании ООП и субъектом? Основное различие в том, что объект моделирует поведение некоторой сущности (т.е. что-то делает), в то время как субъект лишь должен что-то делать и несет за это ответственность. Таким образом, предполагается, что субъект может не справляться со своими обязанностями или вообще их не выполнять. В таком случае требуется исправить этот субъект или заменить его другим.

Ресурсы

Ресурсом будем называть нечто, что может быть использовано субъектом для решения своей задачи. Отметим, что любой субъект может и должен быть ресурсом, т.е. использоваться другими субъектами, иначе он будет бесполезным. С другой стороны, чтобы выполнить свою задачу, субъект не может не использовать ресурсы. Ресурсы можно также считать субъектами, так как они ответственны за выполнение обязанностей, возложенных на них.

Ресурсы характеризуются ограниченным количеством и тем, что изменение состояния ресурса одним субъектом влияет на все другие субъекты, использующие этот ресурс. Поэтому требуется, чтобы субъекты использовали каждый ресурс единообразно, т.е. чтобы использование ресурса одним субъектом не приводило его в неожиданное или некорректное состояние для другого. Для этого вводится понятие контракта.

Контракты

Чтобы решить задачу субъекта, мы должны разложить ее на подзадачи и подобрать субъекты, которые решали бы эти подзадачи. Поэтому для решения своих задач субъект должен взаимодействовать с другими субъектами. *Контракт* регламентирует это взаимодействие, в частности использование общедоступных ресурсов (вообще говоря, бо́льшая часть программных ресурсов является общедоступной). Контракт устанавливает права и обязанности субъектов по отношению друг к другу и к совместно используемым ресурсам, а также соответствие между правами одних субъектов и обязанностями других.

Обязанности – это то, что субъект должен выполнить или обеспечить (напр. обеспечить выделение и инициализацию ресурсов, освободить ресурсы, гарантировать неизменность состояния ресурсов).

Права – это то, на что субъект может рассчитывать (напр. право использовать ресурс, тип передаваемого ресурса, допустимая длительность использования; состояние среды, наличие установленных компонент и т.д.). Права получаются только на основании контракта и должны быть обеспечены соответствующими обязанностями других субъектов.

Чтобы вся система корректно и надежно функционировала, необходимо чтобы каждый субъект выполнял свои функции, а для этого требуется, чтобы его права выполнялись. Чтобы права субъекта выполнялись, нужно, чтобы

1. соответствующие обязанности других субъектов обеспечивали эти права (т.е. были не меньше; напр., если субъект ожидает объект класса C (это его право), а другой субъект обязуется предоставить объект класса S – подкласса C (обязанность), то обязанность обеспечивает право, если S – не подкласс C , то обязанность не обеспечивает право);
2. субъекты выполняли свои обязанности.

Также контракт может уточнять задачу, которую выполняет субъект в контексте данного взаимодействия. Задача, которую решает субъект в контексте некоторого взаимодействия является подзадачей задачи, для решения которой данный субъект разработан.

Возникает вопрос, что же определяет контракт в отношении субъект-ресурс.

В первую очередь, контракт должен задавать тип (класс) или формат передаваемого объекта (здесь и далее понятия «ресурс» и «объект» используются как синонимы). Тип определяет количество памяти (и других ресурсов), которое требуется выделить для создания объекта, множество состояний объекта и его интерфейс. Фактически тип определяет права получателя объекта на его использование и обязанности поставщика на выделение и инициализацию необходимых ресурсов, а также обязанности поставщика или получателя по освобождению ресурсов. Использование типов позволяет обеспечивать и проверять выполнение контракта как статически, так и динамически и широко поддерживается трансляторами.

Кроме того, контракт может задавать характеристики подтипа ресурса. Например, он может указывать допустимые состояния объекта или диапазон допустимых значений при входе в подпрограмму (*предусловие*) и на выходе из подпрограммы (*постусловие*), а также задавать протокол. *Протокол* – подмножество применимых к данному типу операций и, возможно, последовательность их применения, а также возможность изменения состояния ресурса.

Контракт должен указывать время использования объекта, передаваемого по ссылке:

1. локальное использование – объект передается на время выполнения функции (наиболее типичный случай)
2. связанное использование – объект передается на время существования другого объекта (например, итератор действителен, пока существует соответствующий контейнер).
3. событийное использование – получатель может использовать объект до наступления определенного события (напр. до вызова некоторой функции, когда объект удаляется или передается другому пользователю)
4. перманентное использование – получатель может использовать объект

сколь угодно долго (для статических объектов таких, как функции)

5. свободное использование – получатель может пользоваться ресурсом, пока не освободит его (применяется, когда нужно длительное время пользоваться объектом-сервером, который размещается в свободной памяти («куче»)).

6. на заданное время – объект передается на n миллисекунд (встречал описание такого подхода только в печати)

Также контракт должен определять обязанности по выделению и освобождению ресурсов.

Контроль выполнения контракта

Контролировать выполнение контракта можно несколькими способами:

1. встроенными средствами транслятора во время компиляции (в основном это касается контроля преобразования и приведения типов).

2. использование средств языка для проверки корректности ресурса во время выполнения программы (условные операторы, операторы сравнения, встроенные (inline) функции, макросы).

3. использовать дополнительные инструментальные средства.

4. тестирование субъектов (модулей, классов, функций).

1. отладка.

2. проверка выполнения контракта чтением текста программы.

Из распространенных языков программирования концепцию программирования по контракту поддерживает только Eiffel. Eiffel контролирует выполнение *утверждений* – булевских выражений, которые должны быть всегда верны, предусловий, постусловий, *инвариантов* – условий, которые выполняются для объектов класса всегда. В типизированных языках (C++, Pascal, Ada, Java, CLOS) есть проверка типов. В таких языках, как C++ и Java есть как статическая проверка типов (проверка корректности преобразования типов при компиляции) так и динамическая (проверка при выполнении программы соответствия реального типа и приводимого, для классов, имеющих таблицу виртуальных методов). В не типизированных языках (Smalltalk, JavaScript) статической проверки типов нет. В таких языках с объектом связывается некоторое множество применимых к нему методов и свойств, а при вызове метода интерпретатор проверяет, есть ли у используемого объекта соответствующий метод.

В C++ также можно определять протокол const, что весьма полезно и широко используется в стандартной библиотеке. Этот протокол позволяет задать множество функций, гарантирующих неизменность логического состояния объекта. В C++ можно эффективно использовать утверждения (assert), макросы и механизм исключений для проверки инвариантов, предусловий и постусловий.

Распространенные языки программирования и трансляторы не контролируют время и локальность использования ресурса, поэтому программисту остается только документировать время, на которое передается ресурс и, если надо, вручную проконтролировать, что ресурс действительно используется не дольше, чем предусмотрено контрактом. Правда, языки (или их реализации) со сборщиками мусора (Java, C#) автоматизируют «свободное использование» ресурсов и уменьшают вероятность ошибок связанных с неосвобождением или преждевременным освобождением ресурсов за счет размещения передаваемых объектов только в «куче», но в результате существенно страдает производительность программ.

Обработка нарушения контракта

Основная цель использования контрактов – минимизировать количество ошибок и упростить их поиск. Когда мы формализуем права и обязанности элементов программной системы, мы уже при программировании и проектировании избегаем ошибки, к тому же нам проще программировать классы и методы, для которых однозначно определены функции (в смысле «назначение» – термин «функция» в значении «подпрограмма» заменяется синонимами). Контракты также упрощают поиск ошибок, особенно если они поддерживаются инструментальными средствами. Контракты позволяют обнаружить источник ошибок и исправить его. Явное объявление всех ожиданий субъекта связанных с ресурсом

стимулирует сведение к минимуму этих ожиданий, что повышает предсказуемость поведения субъекта и уменьшает вероятность ошибки.

Рассмотрим случай, когда выполнение некоторой части контракта проверяется во время выполнения программы (тестирования). Возникает вопрос, какова должна быть реакция функции на не выполнение условий контракта. Существует несколько вариантов, некоторые из них можно комбинировать:

1. Выдать соответствующее сообщение.
2. Сгенерировать исключение.
3. Прервать выполнение программы.
4 Возвратить сигнализирующее значение из подпрограммы.
5. Проинформировать другим способом ответственный субъект.
6. Попытаться исправить ошибочную ситуацию.

Выбор реакции зависит от того, на каком этапе обнаружена ошибка – во время отладки и тестирования (пункт 6 не подходит), или во время штатного выполнения (пункт 3 не подходит).

Если ошибку выявляет компилятор или другое инструментальное средство, то реакция понятна: выдать соответствующее сообщение, а дело программиста – решить, как ее исправить.

Предположим, мы уже обнаружили нарушение прав одного из субъектов. Тогда нам нужно обнаружить первоисточник ошибки. По контракту мы определяем второй субъект, который отвечает за выполнение нарушенных прав (т.к. контракт определяет соответствие между правами и обязанностями). Далее мы смотрим, были ли натушены права второго субъекта. Если да, то повторяем описанную выше процедуру, иначе он является источником ошибки и требует исправления.

Проектирование

Когда выделяются субъекты и контракты? Субъекты и их функции (задачи) должны определяться на этапе анализа и проектирования, если они относятся к предметной области. Контракты, как часть реализации определяются при проектировании и программировании.

Субъекты могут выделяться на основании вариантов использования[1,4]. Программу можно рассматривать как наибольший субъект, а варианты использования – задачи субъекта. Далее, на основании анализа предметной области, задачи мы разбиваем на подзадачи и связываем их с соответствующими субъектами. Наиболее эффективный метод выделения субъектов – использование CRC-карт (Class-Responsibilities-Cooperation) [2,3], также можно использовать и другие методики объектно-ориентированного проектирования. После выделения субъектов выявляются взаимодействия (кооперации), позволяющие реализовать функции субъектов. На заключительном этапе проектирования и при программировании определяются и уточняются контракты, по которым происходит взаимодействие.

ЗАКЛЮЧЕНИЕ

В работе были определены основные категории в рассматриваемом подходе, даны основные взаимосвязи между этими категориями и показаны возможности применения подхода при разработке программ.

ЛИТЕРАТУРА

1.Гради Буч. Объектно-ориентированный анализ и проектирование: пер. с англ. – М.: Питер, 2000.

2. М. Фаулер, К. Скотт, UML в кратком изложении. Применение стандартного языка моделирования: пер. с англ. – М.: Мир, 1999. – 191 с., ил.

3. Rebecca Wirfs-Brock, Brian Wilkerson, and Lauren Wiener: Designing Object-Oriented Software. Prentice Hall

4. Ivar Jacobson: Object-Oriented Software Engineering: Use Case Driven Approach. Addison-Wesley, 1994

5. Фридман А.Л. Основы объектно-ориентированной разработки программных систем. – М. Финансы и статистика, 2000. – 192 с. ил.