УДК 004.8

Проектирование расширяемой библиотеки классов компонент атрибутов из алгебры кортежей на C#

И. Д. Оверчук, О. Ю. Чередникова
Донецкий национальный технический университет, г. Донецк
e-mail: ivan.overchuk@gmail.com

Аннотация
В статье освещается процесс разработки расширяемой библиотеки классов математических структур под названием компоненты атрибутов. Описание работы сопровождается обоснованиями принятых решений в соответствии с нуждами проекта и рекомендованными паттернами проектирования. Шаблоны объектно-ориентированного программирования позволяют создать многофункциональную библиотеку классов компонент атрибутов с дальнейшей возможностью расширения. Также было освещено применение таких новшеств языка C#, как классы-примеси на основе интерфейсов с реализацией методов по умолчанию.

Общая постановка проблемы

Алгебра кортежей (АК) – это математический аппарат, разработанный Б.А. Куликом в соавторстве в А.Я. Фридманом и описанный в работах [1-6]. Этот математический аппарат относится к классу булевых алгебр и позволяет реализовать алгебраический подход к логическому анализу в системах искусственного интеллекта. В АК, в отличие от формальных систем, где основа – символьные конструкции, в качестве базового выбрано понятие многоместного отношения и предложены обобщения операций алгебры множеств для работы с отношениями, заданными в разных схемах. Основными математическими структурами алгебры кортежей (АК), над которыми производятся операции, подобные операциям над множествами, являются кортежи и системы кортежей.

C-кортеж представляет собой многоместное отношение над типом объектов. C-система – это объединение C-кортежей.

D-кортеж представляет собой объединение одноместных отношений над типом объектов. D-система – это пересечение D-кортежей.

Кортежам и системам кортежей соответствуют такие объекты, как схемы. Схема – это набор свойств, принадлежащих объектам и участвующих в отношении. Эти свойства называются атрибутами кортежей, а их значения (наборы значений) – компонентами атрибутов. И несмотря на то, что в АК кортежи являются элементарными структурами (над компонентами атрибутов операции отдельно не проводятся), на программном уровне вычисления производятся непосредственно именно над компонентами атрибутов. Кортежи представляют собой декартово произведение значений компонент атрибутов.

Компоненты атрибутов бывают пустые, полные и нефиктивные. Пустые и полные описывают соответствующие наборы значений, нефиктивные – непустые и неполные. Под полнотой здесь понимается описание компонентой всех значений из домена атрибута, называемых универсумом.

Иллюстрация математических структур алгебры кортежей приведена на рис. 1.

Постановка задачи

В данной статье освещается процесс разработки расширяемой библиотеки классов компонент атрибутов. Она является частью дипломной работы по программной реализации алгебры кортежей. Целью работы является разработка системы логического вывода в парадигме объектно-ориентированного программирования.

Разработка велась на C# 11.0. Библиотека классов покрыта модульными тестами, которые доказали её работоспособность.

Основные положения реализации библиотеки классов

Классы компонент атрибутов описывают наборы значений, которые затем потребляются классами кортежей, чтобы иметь возможность перечислять сущности со свойствами, имеющими соответствующие значения. Из этого следует, что компоненты атрибутов обязаны реализовывать интерфейс IEnumerable<T>, где T – тип хранимых данных. Родительский абстрактный класс – AttributeComponent<T>, определяющий стандартное поведение при вызове свойств компоненты и методов операций, а также абстрактный метод перечисления хранимых данных.

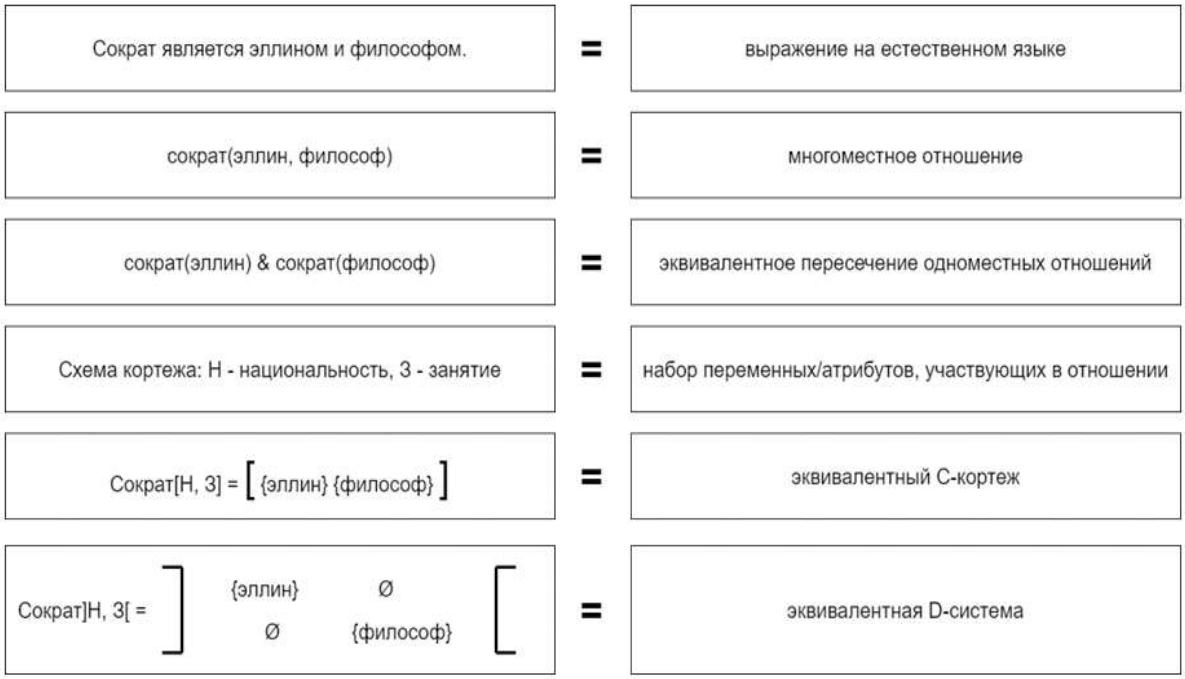

Рисунок 1 – Пример перевода выражения на естественном языке на язык АК

Каждой компоненте соответствуют следующие объекты:

Мощность компоненты: показывает её величину, если это применимо. Пассивно позволяет сравнивать типы компонент (пустая < нефиктивная < полная).

Домен атрибута: компонента должна описывать набор значений из определённого домена. Разделяется компонентами, ему принадлежащими.

Контейнер операторов: вследствие большого разнообразия видов компонент и их комбинаций в операциях набор операторов и сами операторы для каждого класса выделяются в отдельные типы. Разделяется компонентами одного класса.

Фабрика, которая произвела данную компоненту.

Каждый класс компоненты, если это необходимо, имеет статический конструктор, в котором вызывается метод регистрации объектов фабрики и контейнера операторов в качестве ресурсов, разделяемых экземплярами класса. Метод регистрации находится в отдельном классе-регистраторе (рис. 2). Такой механизм необходим для того, чтобы не держать ссылки на фабрики и контейнеры операторов в каждой компоненте. Располагать эти ресурсы в статических свойствах не удастся по причине высокой разветвлённости дерева наследования.

Создание компонент атрибутов следует паттерну «абстрактная фабрика» [7]. Каждый класс фабрики переопределяет методы, связанные с созданием нефиктивных компонент; методы создания пустых и полных компонент являются запечатанными. Каждой фабрике соответствует домен атрибута. Все компоненты, продуцируемые этой фабрикой, принадлежат этому домену. Это решение продиктовано нуждами библиотеки классов кортежей и имеет смысл по причине многофункциональности фабрик компонентов.

Реализация пустых и полных компонент атрибутов

Классы пустой (Empty) и полной (Full) компонент позволяют значительно сократить время выполнения всех операций, поскольку их результаты заведомо известны. Вместо постоянных проверок в коде компонент на предмет пустоты или полноты работа с проходящими проверку идёт как с отдельными нефиктивными компонентами

Реализация нефиктивных компонент атрибутов

Прежде всего стоит отметить, что слова «атрибуты компонентов описывают наборы значений» подразумевают различные методы описания, которые имеют свои цели, преимущества и недостатки. На данный момент библиотека классов нефиктивных (NonFictional) компонент атрибутов содержит следующие:

Итерируемые (Iterable). Такие компоненты создаются на основе перечислений IEnumerable<T>, которые предоставляют удобный инструмент для создания генерирующих последовательностей и конечных автоматов.

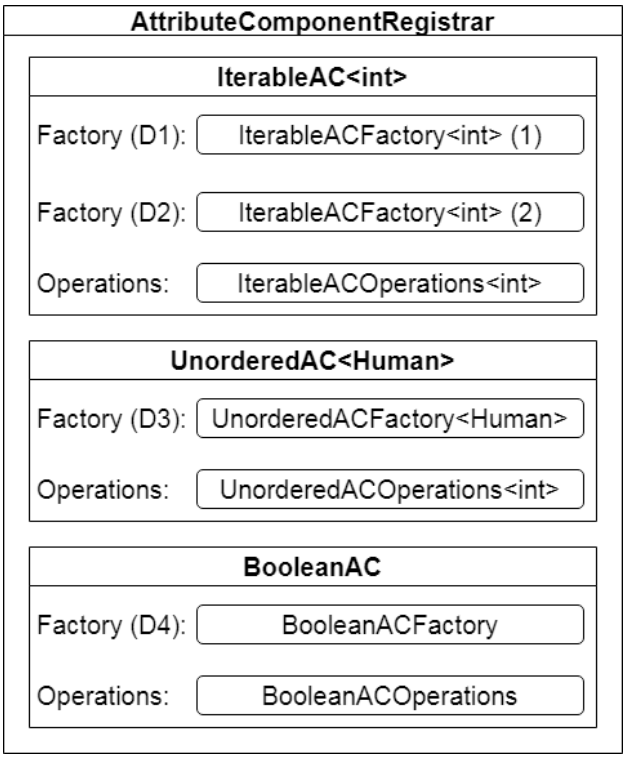

Рисунок 2 – Хранение разделяемых ресурсов в регистраторе классов компонент атрибутов

Преимуществом такого подхода является отсутствие затрат на хранение данных (хранится только код для генерации последовательности) вкупе с минимальными временными затратами на создание объектов. Минус – эффективно использоваться такая структура данных при генерации, а не извлечении из другого перечисления, может только на нессылочных типах данных (структурах), ссылочные будут нагружать сборщик мусора при больших объёмах данных вследствие создания большого числа объектов, подлежащих сборке мусора. Пример использования: генерация числовых рядов.

Неупорядоченные (Unordered). Такие компоненты создаются на основе хеш-таблиц HashSet<T>. Их преимуществом является малое время выполнения операций вида Unordered X Unordered (а также операции дополнения), в общем случае лежащее в пределах [$O(n_1)$, $O(n_1 + n_2)$]. Недостатки те же, что и у хеш-таблиц: затраты на хранение ключей и поиск данных в случае коллизий либо требование к наличию хорошей хеширующей функции. Пример использования: списки персонала организации.

Упорядоченные (Ordered). Такие компоненты создаются на основе любых структур данных, реализующих интерфейс IEnumerable<T>, и компаратора IComparer<T> для упорядочивания данных. Их преимущество – это возможность обходиться почти без затрат на хранение данных благодаря механизму доступа к непрерывной области памяти, реализованному в структурах ReadOnlySpan, ReadOnlyMemory. Для достижения этой цели были разработаны два класса: BufferingOrdered (буферизирующие) и StreamingOrdered (потоковые). Буферизирующие компоненты хранят данные в памяти в виде массива, который представляет собой непрерывную область памяти, и используются в качестве универсума домена атрибута. Потоковые компоненты оптимизируют память при помощи объекта класса ReadOnlySequence – связного списка экземпляров ReadOnlyMemory, областей непрерывной памяти. Операции сравнения вида Ordered X Ordered производятся крайне быстро вследствие упорядоченности ($O(\max(n_1, n_2))$), а прочие операции передают на выход уже отсортированные данные, т. е. сортировать данные в каждом домене нужно только раз – для универсума. Из недостатков можно выделить то, что упорядоченные компоненты эффективно работают лишь при наличии упорядоченного универсума домена и кооперации с себе подобными, а также проблемы потоковых компонент: в худшем случае связный список ReadOnlySequence будет содержать столько же элементов ReadOnlyMemory, сколько элементов представляет компонента (такую редукцию можно устранить путём замены с определённого этапа потоковой компоненты на буферизирующую). Поскольку упомянутые типы для чтения областей памяти обладают довольно скудным программным интерфейсом, был разработан специальный статический класс ReadOnlySequenceHelper для преобразования

ReadOnlySequence в отсортированную по эталону последовательность и наоборот. Пример использования: большие массивы числовых данных.

Булевые (Boolean). Пример специализированной для определённого типа компоненты. Фабрика содержит две константных переменных для экономии ресурсов. Контейнер операций оперирует непосредственно над булевыми значениями. Мощность всегда показывает единицу. Подобное решение можно экстраполировать на компоненту перечислений-флагов (enum с атрибутом типа FlagsAttribute). Плюс такой реализации – большая эффективность.

Фильтрующие (Filtering). Такие компоненты создаются на основе деревьев выражений (expression trees), представляющих предикаты (Predicate<T> или Func<T, bool>). Их назначение – предоставлять функционал SQL/LINQ-запроса WHERE по отношению к компонентам других типов, а также описывать набор данных не правилом генерации, а правилом фильтрации, при этом фильтруются значения из универсума домена. Преимущества следующие: деревья выражений разных фильтрующих компонент можно комбинировать в результате выполнения операций; деревья выражений в потенциале интеллектуально позволяют программе оценивать результат выражения заранее (например, выход за пределы числового ряда) или переводить в форму выражения некоторые последовательности, однако это сопряжено с трудностями реализации. Недостаток: не могут использоваться в качестве универсума домена, поскольку для этого не предназначены. Пример использования: фильтрация универсума целых чисел по признаку чётности.

Также имеются интерфейсы IFiniteEnumerable<T> и ICountable<T>. Первый сообщает лишь о наличии перечисления значений (отсутствует у фильтрующих компонент), второй наследуется от первого и сообщает о наличии у компоненты вычисленного количества хранимых данных (неприменимо к итерируемым компонентам). Эти интерфейсы позволяют обобщить операции на несколько классов компонент.

Реализация мощностей компонент

Класс AttributeComponentPower служит для инкапсуляции операций сравнения двух компонент по количеству хранимых данных (если применимо) и равенства мощности данной компоненты мощности пустой или полной. Первая операция может применяться в алгоритмах операций алгебры множеств, вторая используется фабриками компонент атрибутов для определения необходимости преобразования нефиктивной компоненты в пустую или полную. Соответственно, необходимо проектировать подклассы AttributeComponentPower для каждого подтипа нефиктивной компоненты. Класс AttributeComponentPower возможно реализовать по шаблону «приспособленец» [3], поскольку мощности пустых и полных компонент не требуют внешнего контекста, а для мощностей нефиктивных компонент таким контекстом выступают сами нефиктивные компоненты. Такая реализация значительно уменьшит число хранимых в памяти экземпляров AttributeComponentPower.

Реализация доменов атрибутов

Класс домена атрибута AttributeDomain<T> – это обёртка над классом AttributeComponent<T>. Он может выступать членом любых бинарных операций вместе с другой компонентой, выполнение которых он делегирует свойству Universe – универсуму домена, который представляет собой нефиктивную компоненту. Выделение этого класса необходимо для разделения «полных» нефиктивных компонент и «неполных». Также этот класс позволяет добавить функцию именования доменов (что полезно, например, для хеширования доменов или их идентификации).

Реализация фабрик нефиктивных компонент атрибутов

Классы фабрик наследуются от родительского класса AttributeComponentFactory<T>. Этот класс определяет методы создания пустых и полных компонент, а также шаблонный метод создания нефиктивных: если после создания компоненты её мощность показывает пустоту или полноту, метод возвращает пустую или полную компоненту соответственно.

Фабрики потребляют экземпляры подкласса фабричных аргументов AttributeComponentFactoryArgs, которые содержат необходимые данные (перечисление значений, выражение предиката, мощность компоненты и т. д.). Каждому неабстрактному и использующемуся в коде классу нефиктивной компоненты соответствует такой подкласс (методы создания пустых и полных компонент требуют экземпляр AttributeComponentFactoryArgs, поэтому возможна передача фабричных аргументов для создания нефиктивных компонент). В языке C# отсутствует поддержка множественного наследования, однако его функционал необходим для возможности расширения описываемой библиотеки классов. Поскольку в первую очередь расширяется список классов нефиктивных компонент и требуется поддержка

этих новых классов в других модулях, нельзя статично закодировать их создание и использование, иначе потребуется постоянная рекомпиляция кода, что, в сущности, и противоречит принципу расширяемости. Однако C# предлагает иное, элегантное решение: классы-примеси на основе интерфейсов с реализацией методов по умолчанию. Благодаря этому имеется возможность проектировать классы-примеси фабрик, которые могут создавать нефиктивные компоненты различных видов: упорядоченные, итерируемые, фильтрующие. Данный момент очень важен в библиотеке классов кортежей: при настройке схемы кортежей необходимо указывать фабрики для каждого атрибута.

Подклассы фабрик реализуют интерфейс INonFictionalAttributeComponentFactory<T, TFactoryArgs>. Этот интерфейс определяет метод создания нефиктивной компоненты с определённым подклассом фабричных аргументов. Интерфейсы, наследующие INonFictionalAttributeComponentFactory<T, TFactoryArgs>, добавляют реализацию по умолчанию для этого метода.

Создание кортежей основывается на передаче их фабрикам экземпляров подклассов AttributeComponentFactoryArgs, поэтому следует обеспечивать фабрики компонент всеми нужными возможностями для работы с фабричными аргументами. Однако, так как кортежи принимают экземпляры базового класса AttributeComponentFactoryArgs, кортежи не могут знать, какой интерфейс фабрики соответствует каждому фабричному аргументу, поэтому передача аргумента в фабрику делегируется самому аргументу. Данное поведение также реализуется при помощи примесей.

Реализация контейнеров операторов компонент атрибутов

Компоненты атрибутов поддерживают такие операции алгебры множеств, как отрицание, объединение, пересечение, разность и симметрическую разность, проверку включения одной компоненты в другую, равенства друг другу и включения или равенства. Почти все из этих операций являются бинарными и допускают многие сочетания типов компонент атрибутов, и все эти операции имеют различную реализацию для разных сочетаний. По причине большого числа сочетаний, которое может со временем увеличиваться при расширении библиотеки классов, следует использовать паттерн «Состояние» и вынести все бинарные операции в отдельные классы операторов, которые реализуют паттерн ООП «Посетитель» с дополнительной диспетчеризацией для второго аргумента [7, 8].

Можно выделить две группы бинарных операторов: с немедленным, заведомо известным результатом (в случае пустых и полных компонент) и с вычисляемым, использующим фабрику (для нефиктивных). У первой группы суперкласс называется InstantBinaryOperator, у второй – FactoryBinaryOperator. Их суть одинаковая, отличается лишь сигнатура вызываемых методов: у первой группы в сигнауру входят только два операнда, у второй добавляется фабрика, которая производит результат операции.

Чтобы операции хранились в одном месте, проектируются классы контейнеров операторов. Они также делятся на InstantOperatorContainer и FactoryOperatorContainer, каждому типу контейнеров соответствует тип операторов алгебры множеств. Операторы сравнения реализуются на базе InstantBinaryOperator, поскольку их результатом выступает булево значение. Для каждого подкласса компоненты атрибута создаётся свой подкласс контейнера операторов. FactoryOperatorContainer также содержит фабрику связанной компоненты, поскольку есть возможность избежать её постоянного поиска через регистратор типов компонент.

Есть два уровня операций: «полная/пустая/нефиктивная X полная/пустая/нефиктивная» и «конкретная нефиктивная X конкретная нефиктивная». Методы операций первого уровня реализованы в родительских классах операций вида CrossTypeOperation, методы операций второго уровня реализуются в подклассах суперклассов операторов. Допустимо дополнять их примесями:

Iterable используют интерфейсы вида IFiniteEnumerableBinaryOperation, которые реализуют операции вида IFiniteEnumerable X IFiniteEnumerable и IFiniteEnumerable X Filtering.

Unordered реализуют операции вида Unordered X IFiniteEnumerable (при этом эффективность операций Unordered X Unordered никуда не девается по причине внутренней реализации методов операций над множествами у HashSet<T>) и Unordered X Filtering.

Ordered реализуют операции вида Ordered X Ordered, Ordered X ICountable, а также используют интерфейсы IFiniteEnumerableBinaryOperation.

Boolean реализуют операции вида Boolean X Boolean.

Filtering реализуют операции вида Filtering X Filtering.

Реализация регистратора типов компонент атрибутов

Регистратор типов – это древовидная структура, соответствующая иерархии

регистрируемых типов. Каждому типу компоненты атрибута AttributeComponent<T> соответствует контейнер операторов AttributeComponentOperatorContainer<T> и множество активных доменов AttributeDomain<T>, каждый домен связан с фабрикой AttributeComponentFactory<T>. Регистратор предназначен для централизованного хранения вышеперечисленных объектов, поскольку децентрализованное будет обходиться слишком дорого по памяти. Поиск объектов, соответствующих экземплярам компонент атрибутов, производится следующим образом:

- контейнер операторов ищется лишь исходя из типа компоненты атрибута;
- фабрика ищется по типу и по домену компоненты атрибута.

Пример хранилища регистратора типов компонент атрибутов приведён на рис.3.

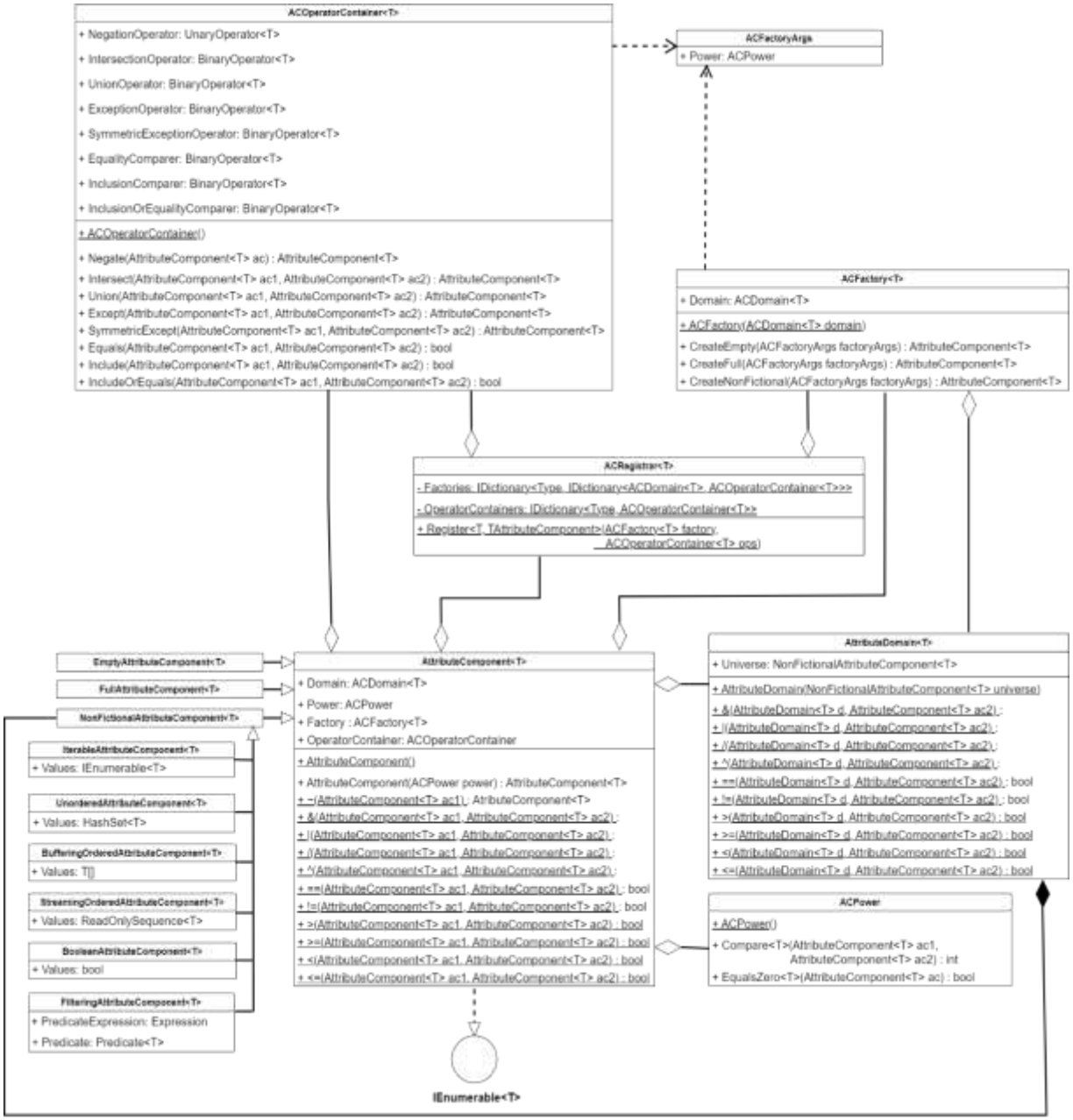

Рисунок 3 – Упрощённая UML-диаграмма классов для компонент атрибутов

При регистрации типов возможно не указывать один или оба объекта для создания (они инициализируются лениво, по мере надобности), в таком случае экземплярам этого типа будут соответствовать:

- контейнер операторов ближайшего типа-предка;
- фабрика ближайшего типа-предка с таким же доменом, что и у экземпляра.

Заключение

Паттерны объектно-ориентированного программирования позволяют создать многофункциональную библиотеку классов компонент атрибутов с дальнейшей возможностью расширения. Начальные этапы работы описаны в [9]. В данной статье освещено применение таких новшеств языка C#, как классы-примеси на основе интерфейсов с реализацией методов по умолчанию.

Литература

1. Кулик, Б. А. Логика и математика: просто о сложных методах логического анализа / Б.А. Кулик; под общ. ред. А. Я. Фридмана. – СПб. : Политехника, 2020. – 141 с. : ил.

2. Кулик, Б. А. Алгебраический подход к интеллектуальной обработке данных и знаний / Б. А. Кулик, А. А. Зуенко, А. Я. Фридман. – СПб.: Изд-во Политехн. ун-та, 2010. – 235 с.

3. Кулик, Б. А. Расширение возможностей логического анализа за счет уточнения интерпретации исчисления предикатов / Информатика и кибернетика. – Донецк: ДонНТУ, 2022. – № 3(29). - С. 5-14.

4. Kulik, B., Fridman, A. Complicated Methods of Logical Analysis Based on Simple Mathematics. – Newcastle upon Tyne: Cambridge Scholars Publishing, 2022. – 195 p.

5. Кулик, Б. А. Исследование противоречий в естественных рассуждениях на примерах метафор и пресуппозиций // Труды Семнадцатой Национальной конференции по искусственному интеллекту с международным участием. КИИ-2019 (21–25 октября 2019 г., Ульяновск, Россия). – Ульяновск: УлГТУ, 2019. Т. 2. – С. 192-200.

6. Кулик, Б. А. Вывод следствий с предварительно заданными свойствами // Системный анализ в проектировании и управлении. Материалы XXV Международной научной и учебно-практической конференции, 13-14 октября 2021 г. – СПб.: ПОЛИТЕХ-ПРЕСС, 2021. Часть 2. – С. 89-97.

7. Гамма, Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р Хелм, Р. Джонсон, Дж. Влиссидес. — СПб.: Питер, 2015. — 368 с.: ил.

8. Мартин, Р. Принципы, паттерны и методики гибкой разработки на языке C# / Р. Мартин, М. Мартин. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 768 с., ил.

9. Оверчук, И. Д. Программная реализация алгебры кортежей с применением методик ORM-технологий / И. Д. Оверчук, О. Ю. Чередникова // Информатика, управляющие системы, математическое и компьютерное моделирование» (ИУСМКМ-2023): сборник трудов XIV международной научно-технической конференции. – Донецк: ДонНТУ, 2023. С. 139-143.

Оверчук И. Д., Чередникова О. Ю. Проектирование расширяемой библиотеки классов компонент атрибутов из алгебры кортежей на C#. *В статье освещается процесс разработки расширяемой библиотеки классов математических структур под названием компоненты атрибутов. Описание работы сопровождается обоснованиями принятых решений в соответствии с нуждами проекта и рекомендованными паттернами проектирования. Шаблоны объектно-ориентированного программирования позволяют создать многофункциональную библиотеку классов компонент атрибутов с дальнейшей возможностью расширения. Также было освещено применение таких новшеств языка C#, как классы-примеси на основе интерфейсов с реализацией методов по умолчанию.*

Ключевые слова: *алгебра кортежей, кортеж, атрибут, домен, логические вычисления, C#, ООП, паттерны проектирования.*

Overchuk I. D., Cherednikova O. Ju. Designing an extensible class library of attribute components from the tuple algebra in C#. *The paper highlights the process of developing an extensible class library of mathematical structures called attribute components. The description of the work is accompanied by justifications of the technological decisions taken in accordance with the needs of the project and recommended design patterns. Object-oriented programming templates allow you to create a multifunctional library of attribute component classes with further extensibility. The application of such innovations of the C# language as impurity classes based on interfaces with the implementation of default methods was also highlighted.*

Keywords: *tuple algebra, tuple, attribute, domain, logical computing, C#, OOP, design patterns.*

Статья поступила в редакцию 12.04.2024
Рекомендована к публикации профессором Павлышом В. Н.