УДК 004.4 DOI: 10.31799/2077-5687-2025-1-57-68

МЕТАУПРАВЛЕНИЕ ФУНКЦИОНАЛЬНОСТЬЮ ИНФОРМАЦИОННЫХ СИСТЕМ. ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ

А. А. Олимпиев
Санкт-Петербургский государственный университет аэрокосмического приборостроения

В статье рассматривается совокупность подходов к разработке программного обеспечения, базирующихся на методологии метауправления функциональностью информационных систем. Описываются результаты анализа приемов разработки программного обеспечения, применяемых на практике. Приводятся основные проблемы, с которыми сталкиваются разработчики, применяющие рассматриваемую методологию. Предлагается иерархия паттернов, которые можно применять для решения данных проблем.

Предлагаемая иерархия паттернов включает в себя три уровня: лингвистический, архитектурный и реализации. Лингвистический уровень определен как основной с точки зрения методологии метауправления функциональностью, остальные — как фундамент и инструменты для организации лингвистического уровня. Иерархия отличается от существующих применением приемов методологии метауправления и учетом особенностей, которые она привносит в общие архитектурные концепции программного обеспечения автоматизированных систем.

Ключевые слова: метауправление функциональностью, языки описания функциональности, паттерны проектирования, архитектурные шаблоны, рецепты разработки программного обеспечения, методы адаптации программного обеспечения без программирования.

Для цитирования:
Олимпиев, А. А. Метауправление функциональностью информационных систем. Паттерны проектирования / А. А. Олимпиев // Системный анализ и логистика. – 2025. – № 1(44). – с. 57-68. DOI: 10.31799/2077-5687-2025-1-57-68.

Введение

Обследование общих тенденций развития современных информационных технологий показало, что одним из наиболее популярных направлений деятельности ведущих разработчиков программного обеспечения является обобщение накопленного опыта решения типовых задач и представление его в виде принципов и способов, которые позволяют быстро найти или повторить решение известных проблем в других проектах.

Полученный опыт обычно описывается в виде пары «проблема-решение», которая получает одно из названий «паттерн», «шаблон» или «рецепт» в зависимости от того, на каком уровне разработки решается проблема.

История концепции «паттернов» начинается со статьи «Язык паттернов» Александера, которую он опубликовал в 1977 году и посвятил архитектуре зданий. Эта статья стала началом большого количества исследований в разных областях знаний, заимствований и обобщений. Предложенный им подход получил широкое распространение и в разработке программного обеспечения. Это привело к тому, что в 2000 году был опубликован «Альманах паттернов» [1], по мнению ведущих специалистов (см. например, [2]) включавший более 1200 паттернов и насчитывающий более 500 различных идей, продолжающих и сейчас развиваться и находить применение в разных направлениях разработки.

Несмотря на многообразие определений термина «паттерн», чаще всего под ним понимается типовой способ или прием, который можно применить для решения известной проблемы. Эта проблема описывается в виде набора характерных признаков, по которым ее можно отличить о других, что существенно упрощает поиск решения. Термин «паттерн» широко известен и занимает устойчивые позиции в различных аспектах разработки программного обеспечения. Так, например, среди всех направлений деятельности и интересов разработчиков, выраженных в названиях книг, можно встретить следующие:

- паттерны проектирования ООП;
- паттерны Kubernetes;

- паттерны проектирования API;
- шаблоны реализации корпоративных приложений;
- паттерны, используемые в машинном обучении;
- паттерны тестирования;
- паттерны микросервисов;
- паттерны предметно-ориентированного проектирования;
- паттерны проектирования на конкретном языке программирования (например, PHP) и многие другие.

Многообразие подобной литературы и статей говорит о общей тенденции, которая мотивирует профессионалов делиться «кристаллизованным опытом» применения и разработки решений типовых задач, полученным на практике. Ни одна серьезная книга по разработке не обходится без описания этого явления. В частности, в [3] архитектурные паттерны предлагается называть «стилями», а также предлагается использовать этот термин при анализе правильности работы команды.

1. Предложения по использованию терминологии

Одним из наиболее сильных толчков для написания этой статьи послужила книга «Чистая архитектура» [4], в которой сформулирована интересная мысль относительно встраиваемого программного обеспечения и микропрограмм, с которыми сталкивается каждый разработчик, например, внедряя SQL в свой код. Автор называет данный подход «антишаблоном», который приводит к непредсказуемым последствиям. В сумме два этих утверждения позволили сформулировать следующие умозаключения:

- утверждение о том, что «смешивание программного обеспечения и микропрограмм — это антишаблон», может быть не вполне верным, если смотреть на этот вопрос с точки зрения методологии метауправления функциональностью (МУФ) информационных систем (ИС), о которой пойдет речь далее;
- применение понятия «антишаблон» может свидетельствовать о наличии проблемы, которая еще не решена, а не о запрете на разработку оригинальных решений, которые обычно пытаются переделать в типовое.

Важно, что устоявшееся негативное отношение разработчиков к «антипаттернам» и «антишаблонам» препятствует активному их исследованию и поиску новых решений известных проблем.

Прежде, чем перейти к описанию сущности методологии метауправления функциональности, целесообразно привести укрупненную классификацию паттернов и отделить термины «паттерн», «шаблон» и «рецепт» друг от друга. Для этого их можно рассматривать с точки зрения уровня системы разработки, на котором они применяются:

- шаблонами предлагается называть архитектурные паттерны, отражающие концепции всей архитектуры и являющиеся фундаментом, на котором строится весь процесс разработки;
- паттерны проектирования — то, что обычно как раз и называют просто «паттернами». Это наборы тех приемов, которые применяются в процессе проработки частных проблем и с которыми сталкиваются разработчики отдельных модулей или компонентов программной системы;
- рецепты — паттерны проектирования, оформленные в виде примера программного кода (реализации) или узкоспециализированного (не обобщенного) решения. Рецепты обычно зависят от выбранной архитектуры, конструкции системы, используемых программных библиотек, особенностей языка программирования и предметной области, поэтому их количество исчисляется тысячами.

2. Методология метауправления функциональностью информационных систем

Методология МУФ ИС изучает особенности технологий создания таких ИС, сопровождение и развитие которых осуществляется без привлечения разработчиков и с минимальным количеством программирования. Также методология изучает способы повышения живучести (долговечности) таких ИС, какие описаны, например, в [5] и особенности, которых описаны ниже.

Исторически, первой монографией, посвященной проблематике МУФ можно считать [6]. Технология применения метаописаний для расширения функциональности ИС, описанная в монографии, на тот момент представляла интерес для многих разработчиков, однако в монографии эти вопросы были впервые систематизированы, что дало толчок многолетнему исследованию, которое продолжается по сей день.

Актуальность проблем создания ИС с МУФ и сегодня подтверждается наличием большого количества статей, в которых описано применение тех или иных приемов разработки (см. например, [7, 8]).

Фундаментальным принципом организации ИС с использованием МУФ является отделение инвариантной (управляющей) составляющей архитектуры от прикладной и представления второй в виде множества компонентов метаинформации (КМИ), расширяющих функциональность ИС за счет их динамического или статического встраивания (инъекции) в инвариантную составляющую различными способами.

Компонент метауправления представляет собой самодостаточный фрагмент описания совокупности новых функций ИС, представленных на языке сверхвысокого уровня, понятном специалисту предметной области. За счет составления комплекта таких описаний пользователь может адаптировать систему под свои нужды. То есть разработке подлежат только КМИ, но не программное обеспечение. Так, например, в статье [5] приведен пример языка программирования онтологий, а в статье [9] приведен пример языка автоматного программирования. Достаточно распространенным приложением методологии МУФ является создание адаптируемых пользовательских интерфейсов (см. например, [10]).

В методологии МУФ используемый для написания КМИ язык называется языком описания функциональности (ЯОФ). Считается, что если ЯОФ достаточно прост для понимания и может быть быстро освоен непрофессионалом, то может быть достигнута очень высокая экономическая эффективность ИС.

Основные проблемы, с которыми сталкивается разработчик ИС с МУФ, можно свести к следующему перечню:

- проблемы выбора ЯОФ, который будет использоваться в системе;
- проблема взаимозависимости многих КМИ;
- проблема контроля версий КМИ;
- проблема распределения обязанностей между модулями системы, в которых реализовано метауправление;
- проблема развития системы (в частности модернизации ЯОФ);
- проблема производительности ИС с МУФ;
- проблема информационной безопасности;
- проблема контроля целостности данных системы.

Выбор способов решения перечисленных проблем представляет собой поиск множества компромиссов, который может вынудить разработчика отказаться от применения методологии из-за сложности и дороговизны разработки такой ИС. Однако составление справочника типовых проблем и их решений (паттернов) может существенно снизить подобные риски.

Даже при поверхностном рассмотрении можно говорить о том, что понятие «метауправление функциональностью ИС» является слишком абстрактным. Поэтому целесообразно рассмотреть, в каких аспектах МУФ себя проявляет:

- рассмотрение с точки зрения архитектуры инвариантной составляющей, позволяет понять, каким образом КМИ встраиваются в систему, что при этом происходит;
- рассмотрение прикладной составляющей системы позволяет говорить о том, от чего зависят КМИ, при каких условиях они актуальны;
- изучение структуры позволяет понять, как КМИ связаны между собой;
- аспект разработки позволяет увидеть, как происходит перемещение КМИ между подпроцессами разработки, какую роль КМИ в каждом подпроцессе занимают, как осуществляется контроль качества и за счет чего достигается требуемый эффект;
- аспект развития указывает на то, как будет модернизироваться система, как будут меняться КМИ, что будет при этом происходить, какие шаги нужно предпринять для предотвращения возможных проблем.

Основным подходом, который отличает методологию от других, принят подход, описанный в [11] и определяющий основные пути развития множества ЯОФ, применяемых в ИС с МУФ.

3. Иерархия паттернов метауправления функциональностью

Применяемые для реализации МУФ в ИС паттерны можно условно разделить на три уровня: лингвистический, архитектурный и реализации. Добавление лингвистического уровня при проектировании и анализе ИС существенно отличает представляемую методологию от других. При этом в данной методологии именно лингвистический уровень является первичным.

На лингвистическом уровне рассмотрение ИС выполняется с точки зрения применения в ней ЯОФ и путей их развития, рефакторинга. В частности, основными решениями, которые можно встретить сводятся к трем основным приемам: использование стандартизованных языков, внедрения новых или унификация существующих [12].

Для принятия решений по выбору ЯОФ может использоваться их классификация по способности менять функциональность ИС. Встречаются следующие виды ЯОФ:

- языки описания архитектуры системы, ярким примером которых является UML и его способность транслироваться в программный код [13];
- языки описания предметной области, такие как IDEF, которые позволяют формировать эффективные модели данных и процессов их перемещения и обработки;
- языки описания взаимодействия компонентов системы, такие как язык описания OpenAPI, SNMP SMI и другие;
- языки описания решений отдельных прикладных задач и формирования прикладных моделей данных, к таким языкам можно отнести отдельные приложения XML, JSON, языки MathML, TeX, WML, CML и многие другие;
- языки описания отдельных процедур и данных такие как шаблонизаторы tpl, twig, нотации Бекуса-Наура, ASN.1 и т. п.;
- универсальные языки программирования общего назначения такие как LUA, Python, TLC и др.;
- встраиваемые языки, например, язык регулярных выражений, SQL-запросов, языки макросов, маркировки и так далее.

Приведенная классификация позволяет понять способы, с помощью которых можно расширяться функциональность ИС: от простого варьирования типами входных данных до изменения структуры системы и ее назначения за счет изменения набора применяемых КМИ.

Однако выбора одного языка, как правило оказывается недостаточно, по той простой причине, что для более-менее сложной системы характерны достаточно большое количество разнообразных вариантов использования [3, 14] и высокая сложность [3, 15]. Поэтому достаточно часто приходится сталкиваться с проблемой комбинирования нескольких языков. Для решения этой проблемы применяются один или несколько из следующих приемов:

- язык A становится надстройкой для языка B — этот прием можно встретить в технологиях шаблонизации и макросов. Работа языка A обычно сводится к формированию варианта КМИ на языке B, зависящего от входных параметров КМИ на языке A;
- язык A встраивается в язык B – примерами этого приема является встраивание регулярного выражения, SQL или другого языка. При этом один из способов взаимодействия сводится к тому, что виртуальная машина языка A передает управление виртуальной машине языка B в процессе работы (поток выполнения языка B является продолжением потока выполнения языка A);
- язык A декорирует средства языка B – этот прием можно встреть в виде так называемых средствах аннотирования и параметризации языковых конструкций. В результате обработки конструкций на языке A на объекты, полученные из языка B, навешиваются дополнительные атрибуты, которые затем используются для валидации, трансформации, сериализации и других процедур, выполняемых над данными посредством рефлексии [16, 17];
- объединение языков — фактически сводится к созданию нового языка, обладающего единым синтаксисом и наиболее удобными для разработчика конструкциями, отобранными из обоих языков. Здесь можно привести в качестве примера элементы языков функционального или декларативного программирования, которые встречаются в императивных языках;
- единый результат трансляции: все используемые языки транслируются в один и тот же язык виртуальной машины. Идея появилась при создании первых компиляторов C [18], одной из ее реализаций является технология .Net. Применение приема встречается там, где требуется популяризация технологии (необходимости привлечения специалистов, обладающих базовыми знаниями разных языков программирования);
- механизм динамически подключаемых библиотек или плаггинов, отличается тем, что результат трансляции языка A динамически подключается к виртуальной машине языка B (встраивается в контекст языка B), такой прием можно встретить, например, в таких языках, как python, LUA, к которым динамически подключаются модули, написанные на языке C или C++;
- иногда встречаются подходы, в которых язык A используется для того, чтобы описать семантику языка B или проверять валидность его алгоритмов, пример декларативное описание инвариантов отдельного алгоритма, которые должны соблюдаться на входе и выходе (см. например, [19]).

Для методологии МУФ также представляют интерес специальные паттерны лингвистического подхода, которые используются для организации языка. К таким паттернам можно отнести грамматики, идиомы и стереотипы, на которых базируются синтаксические, концептуальные и семантические особенности языка [20, 21].

Описанные выше паттерны образуют слой лингвистической архитектуры и строятся на базе инвариантного ядра ИС, в котором происходит перемещение и трансформация готовых КМИ.

На уровне архитектуры инварианта, втором уровне иерархии, можно выделить следующие основные паттерны:

- «метамодель» — сверхобобщенная модель данных, позволяющая с помощью базовых категорий формулировать множество прикладных конкретных моделей и представляющая собой базовый язык инварианта ИС (язык реализации). Метамодель является неотъемлемым элементом парадигмы МУФ и определяет грамматику ЯОФ;
- «реестр метаописаний» — универсальное хранилище КМИ, которое используется в архитектурах ИС с централизованным МУФ. Данный паттерн является достаточно распространенным и используется в системах с высокой динамикой предметной области, либо в случае ограниченности срока жизни отдельного КМИ. Актуализация КМИ может выполняться разными способами, так, например, в [5, 22] описан формализм языка запросов, зависящего от используемой конфигурации (онтологии), а в [7, 23] приводится формализмы, базирующиеся на объектно-ориентированном подходе;
- «управление версиями КМИ» — паттерн предназначенный для автоматизации процессов своевременной актуализации функциональности. Данный паттерн отличается от похожего и описанного в [24] тем, что обновлять необходимо не саму модель, а КМИ, на основании которых очередная версия модели будет синтезирована;
- «генератор модели» осуществляет формирование прикладной модели данных X из описаний, представленных на языке L_1. Полученная модель будет решать конкретную прикладную задачу F(x);
- «трансформатор модели» осуществляет преобразование данных модели X в данные модели Y, алгоритм трансформации задается с помощью КМИ на языке L_2;
- «генератор языка» осуществляет преобразование данных модели X в формат L_Z внешней системы (например, формат сообщений, которые отправляются по сети, или разметку страницы для печати, вывода на экран или передачи WEB-браузеру). Как осуществлять преобразование X в L_Z задается с помощью КМИ на языке L_3;
- «адаптер» решает обратную задачу относительно «генератора языка» — преобразует данные, поступающие в формате L_Y внешней системы в данные модели X. Как осуществлять преобразование L_Y в X задается с помощью КМИ на языке L_4;
- «трансформатор языка» применяется в тех ситуациях, когда данные в формате внешней системы L_W нельзя сразу отобразить на модель X, а необходимо отправить по сети или сохранить в некоторый внутренний формат обмена данными L_4, либо наоборот сформировать пакет данных в формате языка L_W из внутреннего формата L_4. Для описания алгоритма трансформации применяется соответствующий КМИ, представленной на языке L_5;
- «агрегатор» применяется в тех ситуациях, когда нужно собрать модель X или язык L_{N+1} из фрагментов, представленных на языках L_1...L_N, либо разложить данные X или язык L_{N+1} на языки L_1...L_N. Функциональность таких элементов системы, как правило, полностью задается с помощью программирования ввиду сложности реализации и тестирования;
- «генератор протокола» позволяет синтезировать программную логику сетевого взаимодействия двух или более узлов сетевой информационной системы при условии, что они оба обмениваются данными с помощью внутреннего языка L_7. Протокол как правило задаемся с помощью языка описания API и комбинируется с языками описания моделей данных узлов в сумме образующие КМИ на языке L_8.

Важным отличием перечисленных паттернов заключается в том, что алгоритмы их работы заложены не полностью, а доопределяются набором КМИ, представленными на одном из языков (L_1-L_N). На рисунках 1-3 иллюстрируется отдельные паттерны и их комбинации.

На рисунке 1 демонстрируется особенность паттерна «метамодель», суть которого сводится к тому, что каждый КМИ наделяет ИС способностью порождать модели определенного типа. При этом, как видно из рисунка, один КМИ дает возможность описывать множество комбинаций различных объектов и отношений, но каждый отдельный объект и отношение могут быть описаны только одним способом. Для того, чтобы задать новый способ описания объектов, создается новый КМИ. Функционал Ψ задает базовые правила порождения моделей посредством реализованного формализма.

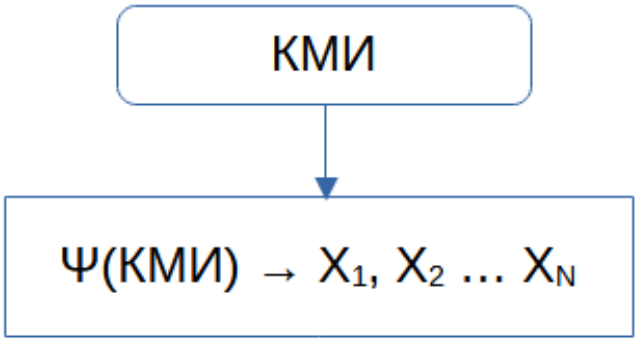

Рис. 1. Паттерн «Метамодель».

На рисунке 2 приведен пример применения паттернов «реестр метаописаний» и «управление версиями КМИ».

На первый взгляд реализация данного паттерна кажется достаточно простой, однако здесь стоит обратить внимание на часто возникающую проблему. Изменение версии КМИ обычно приводит к тому, что модели, полученные с помощью предыдущей версии, могут потерять актуальность, поэтому рекомендуется также применять технологии миграций, которые позволят управлять изменениями данных при переходе между версиями КМИ.

На рисунке 3 продемонстрирована концепция агрегирующего КМИ, который описывает всевозможные архитектуры ИС и содержит в себе информацию о типах узлов, моделях, протоколах информационного обмена, ролях, правилах преобразования данных и т. п.

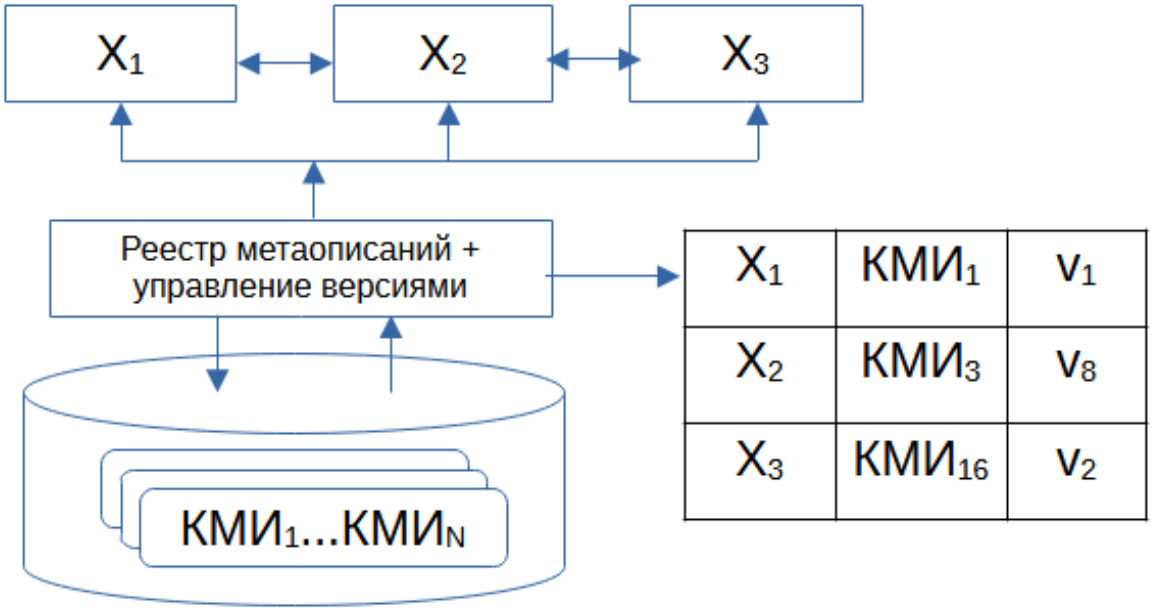

Рис. 2. Паттерны «реестр метаописаний» и «управление версиями»

При порождении конкретной архитектуры (в данном случае состоящей из двух узлов) из агрегирующего КМИ выделяются и параметризуются фрагменты, каждый из которых описывает элемент системы:

- $КМИ_1$ и $КМИ_6$ описывают как должны порождаться соответственно модели X_2 и X_1;
- $КМИ_2$ описывает, как из протокольного сообщения (Protocol data unit, PDU) получить данные для модели X_2;
- $КМИ_5$ описывает преобразование из X_1 в PDU;
- $КМИ_3$ и $КМИ_4$ описывают, как должен работать протокол (порядок обмена сообщениями).

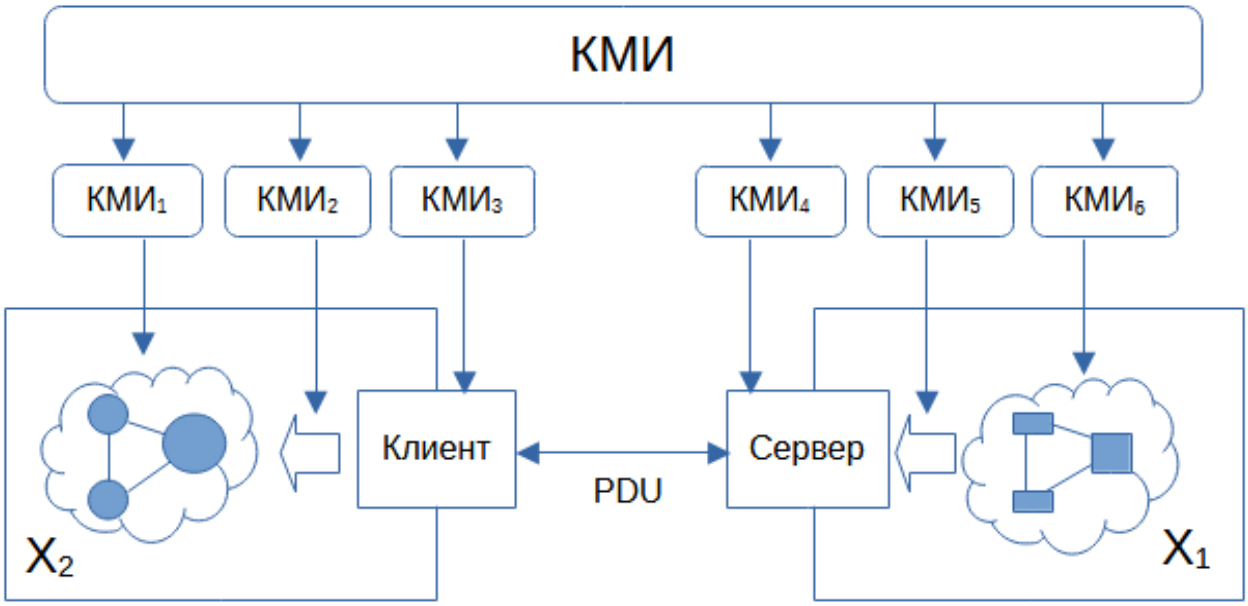

Рис. 3. Генерация клиент-серверного приложения с помощью КМИ

Приведенный пример отдаленно напоминает концепцию «инфраструктура как код» (IaC) [25], которая становится все более популярна в DevOPs, и может лечь в основу технологии быстрого развертывания типовой микросервисной архитектуры любой сложности за счет создания эталонного метаописания. В статье [26] можно найти другой пример реализации (монолитной архитектуры) с помощью данного паттерна.

Как видно из представленного перечня паттернов, количество языков, которые могут одновременно встречаться в системе, достаточно большое, поэтому и сама система является сложной как с точки зрения реализации, так и с точки зрения понимания ее работы и сопровождения. Это является серьезным препятствием в массовом распространении подобных систем и быстрому их развитию.

В современных статьях паттерн «метамодель» часто обозначается как вычислительный формализм или просто формализм, и как правило основан на одной из известных технологий синтеза моделей. Так, например, встречаются формализмы, основанные на объектно-ориентированном подходе [27], на онтологическом подходе [5], функциональном подходе [9] и другие.

Следует отметить статью [28], где описаны отдельные приемы, с помощью которых можно решать проблемы распространения связанных между собой КМИ. Поскольку применение приемов МУФ очень популярно в WEB-технологиях из-за их особенностей, то для методологии МУФ представляет интерес приемы разработки, которые там применяются. В частности, в статье [28] описывается прием к управлению фрагментами JS, который с точки зрения МУФ классифицируется как универсальный ЯОФ общего назначения, а конкретные модули JS — это КМИ, написанные на этом языке.

Третий уровень иерархии образуют паттерны реализации, которые включают в себя по большей части широко известные паттерны проектирования и рецепты. Наиболее важные с точки зрения метауправления можно разбить на две достаточно большие группы: паттерны трансляции и паттерны времени выполнения.

К паттернам этапа трансляции относятся: макрос, шаблонизатор, конкатенатор, декоратор, грамматика, абстрактное синтаксическое дерево, транслятор, линковщик.

К паттернам времени выполнения можно отнести: рефлексия, менеджер плаггинов, переключатель конфигураций, генератор отображений, реестр КМИ, интерпретируемый объект, менеджер контекста, интерпретатор, контекст, метаклассы и метаобъекты.

Паттерны реализации в основном базируются на обобщенных паттернах «Банды четырех» [29], таких как «стратегия», «алгоритм», «компоновщик», «итератор», «декоратор» и других, но имеют свои особенности.

Важное место с точки зрения современных технологий разработки и фрейморков занимает паттерн «Инъекция зависимостей», ценность и удобство применения которого отмечены в [30].

Заключение

Вопросы, освященные в рамках настоящей статьи, рассматриваются и другими исследователями. Например, в [31, 32] предлагаются готовые рецепты, которые можно обобщить и повторно использовать. Однако, основная масса работ посвящена частным приложениям МУФ, в которых отсутствуют описания парадигм разработки и паттернов, поэтому данная статья является актуальной и своевременной.

Большинство разработчиков склоняются к применению стандартизованных готовых решений и в их статьях в основном делается акцент на онтологические и объектно-ориентированные формализмы. В то же время все большую популярность набирает функциональная модель программирования, что может существенно повлиять на дальнейшее развитие технологий.

Также хотелось бы обратить внимание на то, что методология МУФ не имеет четких границ и может рассматриваться как «аккумулирующий носитель» приемов и способов определенного вида, которые передаются от одной информационной технологии в другие. При этом методология обладает уникальностью аспектов рассмотрения включает в себя совокупность самостоятельных, самодостаточных научных подходов.

В качестве вывода можно сформулировать следующее утверждение: методология МУФ предлагает альтернативную относительно [4] и [33] технологию, позволяющую получить «чистую архитектуру» ИС, но не в виде ядра и набора плаггинов, которые реализуют детали системы, а в виде инварианта, который интерпретирует КМИ или транслирует их в расширения функциональности, встраиваемые в ядро системы.

Еще одним достоинством методологии является то, что она позволяет не только сконструировать новую ИС с МУФ, но и выполнять анализ, модернизацию, рефакторинг и оптимизацию существующих ИС [34].

СПИСОК ЛИТЕРАТУРЫ

1. *Rising L.* Pattern Almanac / Linda Rising. – Boston: Addison-Wesley, 2000. – 336 p.
2. *Ларман К.* Применение UML 2.0 и шаблонов проектирования / К. Ларман; пер. с англ. А. Ю. Шелестова. – М.: ООО «И.Д. Вильямс», 2018. – 736 с.
3. Современный подход к программной архитектуре: сложные компромиссы / Н. Форд [и др.] – СПб.: Питер, 2023. – 480 с.
4. *Мартин Р.* Чистая архитектура. Искусство разработки программного обеспечения. / Р. Мартин. – СПб.: Питер, 2023. – 352 с.
5. *Грибова В. В.* Разработка решателей задач на основе управляющих графов для систем с базами знаний / В. В. Грибова , Ф. М. Москаленко, В. А. Тимченко , Е. А. Шалфеева // Программная инженерия. – 2021.– Том 12, № 3. – С. 115-126.
6. *Шерстюк Ю. М.* Основы метауправления функциональностью в информационных системах / Ю. М. Шерстюк. – СПб.: СПИИРАН, 2000. – 155 с.
7. *Духовенский С. Е.* Разработка репозитория метаданных на основе объектно-ориентированной логической модели для оценки качества данных / С. Е. Духовенский // International Journal of Open Information Technologies. – 2024. – Т.12, №4. – С. 60-67.
8. *Шитов А. В.* Программный комплекс механизма управления доступом на основе риск-ориентированной атрибутивной модели. / А. В. Шитов, Н. Е.Стельмах, Ш. Г. Магомедов // International Journal of Open Information Technologies – 2024. – Т.12, № 6. – С. 133-142.
9. *Шелехов В. И., Тумуров Э. Г.* Технология автоматного программирования на примере программы управления лифтом / В. И. Шелехов, Э. Г. Тумуров // Программная инженерия. – 2017. – Том 8, № 3. – С. 99-110.
10. *Степанов М. Ф.* Адаптивный пользовательский интерфейс системы автоматизированного анализа и синтеза алгоритмов управления / М. Ф. Степанов, А. М. Степанов // Программная инженерия. – 2018. – Том 9, № 3. – С. 109-121.
11. *Олимпиев А. А.* Способ управления жизненным циклом программного изделия // Информационная безопасность регионов России (ИБРР-2021): XII Санкт-Петербургская межрегиональная конференция. – СПб., 2021. 94-95 с.
12. *Олимпиев А. А.* Способ снижения разнообразия языков описания функциональности // Актуальные проблемы инфотелекоммуникаций в науке и образовании: сб. науч. ст. XI Международная научно-техническая и научно-методическая конференции. – СПб., 2022. – Т. 2. – С. 450-454.
13. *Леоненков А. В.* Самоучитель UML 2 / А. В. Леоненков. – СПб.: БХВ-Петербург, 2007. – 576 с.

14. *Калянов Г. Н.* О теории бизнес-процессов / Программная инженерия. – 2018.– Том 9, № 3. – С. 99-108.
15. *Косяков А.* Системная инженерия. Принципы и практика / А. Косяков, У. Свит и др.; пер. с англ. под ред. В. К. Батоврина. – М.: ДМК Пресс, 2017. – 624 с.
16. *Шилдт, Г.* Java 8. Полное руководство. 9-е изд. / Герберт Шилдт; пер. с англ. И. В. Берштейна. – М.: ООО «И.Д. Вильямс», 2015. – 1376 с.
17. *Смит Дж. П.* Entity Framework Core в действии / П. Дж. Смит; пер. с англ. Д. А. Беликова. – М.: ДМК Пресс, 2022. – 690 с.
18. *Страуструп Б.* Дизайн и эволюция C++ / Б. Страуструп; пер. с англ. А. А. Слинкин. – М.: ДМК Пресс, 2006 – 448 с.
19. *Чушкин М. С.* Система дедуктивной верификации предикатных программ / М. С. Чушкин // Программная инженерия. – 2016. – Том 7, № 5. – С. 202-210.
20. Компиляторы: принципы, технологии и инструментарий, 2-е изд. / Альфред В. Ахо, Лам [и др.]; пер. с англ. И. В. Красикова. – М.: ООО «И.Д. Вильямс», 2008. – 1184 с.
21. *Кауфман В. Ш.* Языки программирования. Концепции и принципы / В. Ш. Кауфман. – М.: ДМК Пресс, 2010. – 464 с.
22. *Корзун Д. Ж.* Формализм сервисов и архитектурные абстракции для программных приложений интеллектуальных пространств / Программная инженерия. – 2015. – № 2. – С.3-12.
23. *Дородных Н. О.* Использование диаграмм классов UML для формирования продукционных баз знаний / Н. О. Дородных, А. Ю. Юрин // Программная инженерия. – 2015. – № 4. – С.3-9.
24. *Лакшманан В.* Машинное обучение. Паттерны проектирования / В. Лакшманан, С. Робинсон, М. Мунн; пер. с англ. – СПб.: БХВ-Петербург, 2022. – 448 с.
25. *Митра Р.* Микросервисы. От архитектуры до релиза / Р. Митра, И. Надареишвили. – СПб.: Питер, 2023. – 336 с.
26. *Ицыксон В. М.* LibSL – язык спецификации компонентов программного обеспечения / В. М. Ицыксон // Программная инженерия. – 2018. – Том 9, № 5. – С. 209-220.
27. *Олимпиев А. А.* Вычислительный формализм синтаксически вариантной системы оперативно-технического мониторинга гетерогенных сетей связи / Региональная информатика и информационная безопасность. Сборник трудов. Выпуск 1. – СПб., 2015. – С. 57-62.
28. *Орлов Д. А.* Архитектура связанных JavaScript микрокомпонентов для крупных веб-проектов. / Д. А. Орлов, Р. А. Карелова // Программная инженерия. – 2021. – Том 12, № 2. – С. 74-81.
29. *Гамма Э.* Паттерны объектно-ориентированного проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – СПб.: Питер, 2023. – 448 с.
30. *Мартин Р.* Чистый код: создание, анализ, рефакторинг. Библиотека программиста / Р. Мартин. – СПб.: Питер, 2010. – 464 с.
31. *Тельнов В. П., Коровин Ю. А.* Программирование графов знаний, рассуждения на графах. / В. П. Тельнов, Ю. А. Коровин // Программная инженерия. – 2019. – Том 10, № 2. – С. 59-68.
32. *Грибова В. В.* Базовая технология разработки интеллектуальных сервисов на облачной платформе IACPaaS. Часть 2. Разработка агентов и шаблонов сообщений. / В. В. Грибова, А. С. Клещев, Д. А. Крылов, Ф. М. Москаленко, В. А. Тимченко, Е. А. Шалфеева // Программная инженерия. – 2016. – Том 7, № 1. – С. 14-20.
33. *Ричардс М.* Фундаментальный подход к программной архитектуре: паттерны, свойства, проверенные методы / Марк Ричардс, Нил Форд. – СПб.: Питер, 2024. – 448 с.

34. *Олимпиев А. А.* Методика синтеза системы оперативно-технического мониторинга с метауправлением функциональностью // Актуальные проблемы инфотелекоммуникаций в науке и образовании: материалы VII Международной научно-технической и научно-методической конференции / Под. ред. С. В. Бачевского; сост. А. Г. Владыко, Е. А. Аникевич. – СПб: СПбГУТ. – 2018. – Т. 2. – С. 505-510.

ИНФОРМАЦИЯ ОБ АВТОРЕ

Олимпиев Алексей Александрович
Канд. техн. наук
Санкт-Петербургский государственный университет аэрокосмического приборостроения
Россия, 190000, Санкт-Петербург, ул. Большая Морская, д.67, лит. А
E-mail: olimpiev@guap.ru