Санкт-Петербургский Государственный Университет

Математико-механический факультет

Кафедра информатики

Зиненко Ольга Игоревна

Систематизация и анализ паттернов проектирования на основе стандартов теории решения изобретательских задач

Дипломная работа

Допущена к защите.

Зав. кафедрой:
д. ф.-м. н., проф. Косовский Н.К.

Научный руководитель:
ст. п. Одинцов И.О.
ст.н.с. Рубин М.С.

Рецензент:
Косовский Н.К.

Санкт-Петербург
2010

Saint-Petersburg State University

Faculty of Mathematics and Mechanics

Computer Science Department

Zinenko Olga Igorevna

Systematization and analysis of design patterns on basis of standards of the Theory of inventive problem solving

Graduate work

Allowed to defence.

Head of a chair:
Professor, Doctor N. K. Kossovski

Scientific adviser:
Senior lecturer I. O. Odintsov
Senior staff scientist M. S. Rubin

Reviewer:
N. K. Kossovski

Saint-Petersburg
2010

Содержание

Введение

Компьютерная индустрия все стремительнее развивается, возрастает производительность систем, возможности обработки данных. С одной стороны - труд программиста облегчается, с другой - все больше требований предъявляется разработке программных продуктов. Конкуренция на рынке программного обеспечения ужесточает рамки, в пределах которых вынуждены работать разработчики: программные продукты должны создаваться за минимально возможное время при максимальном качестве конечного результата. Хотя зачастую эти условия могут противоречить друг другу. В каждом конкретном случае приходится искать "золотую середину" - оптимальное сочетание критериев, при соблюдении которых продукт будет востребован на рынке.

Существует инструмент, при правильном использовании которого разработчик получает возможность применять проверенные временем, испытанные на практике приемы решения наиболее часто встречающихся задач проектирования. А значит, при создании собственной архитектуры можно оптимальным способом расходовать время и поддерживать высокий уровень качества, если там, где это возможно, использовать подходящие удачные и гибкие решения. Этот инструмент - паттерны (шаблоны) проектирования.

В то же время, многие проблемы, с которыми сталкиваются программисты, можно назвать изобретательскими, то есть в явном или скрытом виде содержащими в себе противоречие. Например, необходимо повысить скорость работы приложения, но тогда возрастет объем потребляемых ресурсов. Теория решения изобретательских задач (ТРИЗ) - это технология поиска решений для конкретных задач, изначально разрабатываемая для решения задач технического характера. Но область решаемых задач может быть существенно расширена, в нее можно включить сферу программирования, таким образом иметь возможность применять весь инструментарий ТРИЗ для задач программирования. В том числе и систему

стандартов, представляющую собой правила, по которым должна быть преобразована исходная задача для получения решения.

Можно найти сходство между паттернами проектирования и стандартами ТРИЗ, оба эти инструмента предоставляют возможность найти готовое оптимальное решение наиболее распространенных задач из области проектирования и технической области соответственно.

На основе сходства можно создать единый алгоритм применения стандартов и паттернов, призванный облегчить процесс проектирования и перепроектирования программного продукта.

Возможность расширения инструментария разработчика с помощью единого алгоритма для стандартов ТРИЗ и шаблонов является основным фактором **актуальности** работы.

Объектами исследования в данной работе являются паттерны проектирования и система стандартов ТРИЗ.

Предмет исследования - единый алгоритм решения задач программирования на основе применения стандартов ТРИЗ и паттернов проектирования.

Целью данной работы является создание единого алгоритма решения задач проектирования и его программная реализация.

Для достижения основной цели были поставлены и решены следующие **задачи**:

1. Анализ литературы по теме работы.
2. Создание картотеки задач из области программирования.
3. Адаптация стандартов ТРИЗ для области программирования.
4. Анализ алгоритма применения стандартов ТРИЗ.
5. Анализ, систематизация и перевод паттернов проектирования на общесистемный язык.
6. Создание единого алгоритма применения стандартов и паттернов.
7. Разработка программного приложения, реализующего работу единого алгоритма.

Анализ литературы выявил, что данная **работа не имеет** общедоступных **аналогов**, исследование проводится на стыке двух областей и результатом является возможность применения существующего инструментария для объединенного спектра задач.

Глава 1. Постановка задачи и обзор основных концепций

1.1 Описание проблемной области

Шаблон проектирования программы – это архитектурная конструкция, представляющая собой оптимальное решение некоторой проблемы проектирования в рамках конкретных задач, также включающее в себя рекомендации по применению этого решения в различных ситуациях. Паттерн проектирования обязательно имеет общеупотребимое наименование. Правильно выбранный и сформулированный паттерн позволяет пользоваться удачно подобранным решением задачи проектирования программы множество раз с учетом конкретных требований. Важно, что при работе с шаблонами необходимо построить адекватную модель предметной области, то есть рассматривать ее в некотором аспекте, с точки зрения которого выделяются существенные свойства и параметры, а несущественные не учитываются.

Паттерны имеют достоинства и недостатки. Среди преимуществ можно выделить следующие:

1) модель системы, построенная в терминах паттернов проектирования, фактически является структурированным выделением тех элементов и связей, которые значимы при решении поставленной задачи;

2) модель, построенная с использованием паттернов проектирования, проста и наглядна в изучении, тем не менее она позволяет глубоко и всесторонне проработать архитектуру разрабатываемой системы;

3) применение паттернов проектирования повышает устойчивость системы к изменению требований и упрощает неизбежную последующую доработку системы;

4) совокупность паттернов проектирования, по сути, представляет собой единый словарь проектирования, который, будучи унифицированным средством, незаменим для общения разработчиков друг другом.

Существенные недостатки:

1) когда количество шаблонов возрастает, превышая критическую сложность, исполнители начинают игнорировать шаблоны и всю систему, с ними связанную;

2) применение шаблонов из справочника, без осмысления причин и предпосылок выделения каждого отдельного паттерна, замедляет профессиональный рост разработчика, так как подменяет творческую работу механическим использованием шаблонов.

Однако, наличие недостатков не ставят под сомнение пользу от применения паттернов, а лишь указывает на необходимость их тщательного изучения и применения только в подходящих ситуациях.

Теория решения изобретательских задач (ТРИЗ) - это технология поиска идей для решения конкретных задач, основанная на законах развития технических систем. Основная идея ТРИЗ: технические системы возникают и развиваются по определенным законам, эти законы можно познать и использовать для сознательного (в отличие от метода проб и ошибок) решения изобретательских задач.

Основными элементами ТРИЗ являются:

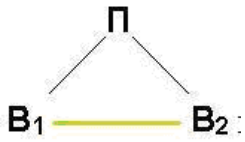
1) законы развития технических систем - наиболее общие статистические закономерности и тенденции развития техники, выявленные в результате анализа патентного фонда и истории развития техники;

2) АРИЗ - алгоритм решения изобретательских задач - комплексная программа алгоритмического типа, основанная на законах развития технических систем и предназначенная для анализа и решения изобретательских задач;

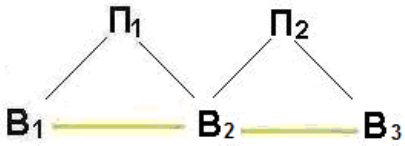
3) вепольный анализ (структурный вещественно-полевой анализ) позволяет представить структурную модель исходной технической системы, выявить ее свойства, с помощью специальных правил преобразовать модель задачи, получив тем самым структуру решения, которое устраняет недостатки исходной задачи. Это специальный язык формул, с помощью

которого легко описать любую техническую систему в виде определенной (структурной) модели. Построенная таким образом модель преобразуют по специальным правилам и закономерностям, получая структурное решение задачи.

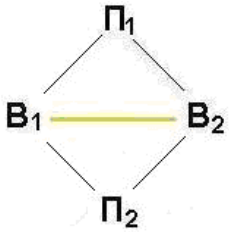
Любой объект представляется в виде вещества и обозначается В, а любое взаимодействие в виде поля и обозначается П. Тогда веполь может быть представлен в виде формулы:



Цепной веполь образуется при превращении одной из частей веполя в независимо управляемый веполь:



Если в веполь ввести второе поле взаимодействия между веществами образуется двойной веполь:



Комплексный веполь образуется при введении добавок или присоединении внешних веществ к одному из веществ веполя:

4) таблица устранения технических противоречий, в которой противоречия представляются двумя конфликтующими параметрами. Для каждого сочетания параметров предлагается использовать один или несколько из сорока приемов устранения противоречия. Приемы

сформулированы и классифицированы на основе статистических исследований изобретений;

5) стандарты ТРИЗ - правила, указывающие, как должна быть преобразована исходная система для решения типовой задачи. Их систематизированная совокупность называется системой стандартов ТРИЗ. Система стандартов на решение изобретательских задач была разработана Г.С.Альтшуллером, изобретателем, писателем, автором теории решения изобретательских задач - теории развития технических систем. В своем развитии система стандартов прошла несколько этапов, в 1985 году разработана система 76 стандартов, состоящая из 5 классов, где каждый из классов включает подклассы и сами стандарты:

1) Построение и разрушение вепольных систем.
2) Развитие вепольных систем.
3) Переход к надсистеме и на микроуровень.
4) Стандарты на обнаружение и измерение.
5) Стандарты на применение стандартов.

1.2 Постановка задачи

Так как паттерны предоставляют простые и гибкие решения часто повторяющихся задач проектирования, то вместо того, чтобы каждую новую задачу решать с нуля, можно повторно использовать удачные, проверенные и динамичные решения. Благодаря высокому уровню абстракции, присущему паттернам, можно успешно решать разнообразные по сложности задачи проектирования, также знание шаблонов способствует лучшему взаимопониманию между разработчиками, что является важным фактором процесса разработки.

Таким образом, шаблоны являются мощным инструментом для разработчиков, занимающихся объектно-ориентированным программированием, польза от их уместного и профессионального

использования не вызывает сомнений. Поэтому одной из основных задач данной работы является создание системы указаний для выбора паттернов.

Паттерны проектирования имеют много общего со стандартами теории решения изобретательских задач. Шаблоны оформлены на основе повышения степени абстрагирования и обобщения решаемых задач программирования, аналогичные подходы были использованы при создании системы стандартов ТРИЗ. Стандарты ТРИЗ могут применяться для решения не только технических, но и задач программирования, то есть вероятно, что одни и те же задачи можно решать и с помощью паттернов проектирования, и с помощью системы стандартов ТРИЗ.

Спектр задач программирования, которые можно решить, используя стандарты ТРИЗ, очевидно много шире относительно задач, решаемых с помощью паттернов. Алгоритм применения шаблонов проектирования будет являться лишь частью алгоритма решения задач с помощью стандартов. Большое внимание в данной работе будет уделяться алгоритму решения задач из области программирования с помощью стандартов ТРИЗ. В связи с чем, с одной стороны, необходимо проделать работу по адаптации теории решения изобретательских задач для области программирования, с другой - перевести паттерны на общесистемный язык для включения в общий алгоритм решения задач из области программирования.

В рамках данной работы планировалось создать картотеку задач проектирования для алгоритма применения паттернов и стандартов ТРИЗ, включающую задачи двух уровней:

1) задачи, элементами которых являются классы и объекты с точки зрения объектно-ориентированного программирования (ООП);

2) задачи, элементами которых будут объединения классов и объектов ООП, рассматриваемые как неделимые объекты.

Конечным результатом будет являться программное приложение, реализующее работу алгоритма решения задач с помощью стандартов ТРИЗ и паттернов проектирования.

1.3 Причины возникновения проблемы и ее актуальность

В программировании, как и в изобретательстве, часто приходится решать изобретательские задачи. Даже несложные на первый взгляд задачи проектирования в процессе поиска решения или даже в самой формулировке обнаруживают противоречия, а значит, могут быть отнесены к разряду изобретательских, для решения которых существует множество инструментов ТРИЗ.

Инструменты ТРИЗ хорошо классифицированы и алгоритмизированы, в то же время одним из недостатков шаблонов проектирования является отсутствие четких указаний, как нужно преобразовать исходную задачу, чтобы решением стал один или группа паттернов. Несмотря на то, что в классической литературе по паттернам для каждого из шаблонов описаны их назначения, мотивации и применимость, проблема алгоритмизации процесса выбора конкретного шаблона для конкретной задачи является актуальной. Интуитивный поиск, поиск, основанный на личном опыте, или простой перебор паттернов не могут являться ее решением. Необходимо создать алгоритм, который бы являлся своего рода поисковой системой для выбора подходящего паттерна или группы паттернов для конкретной задачи.

1.4 Краткий обзор литературы, посвященной проблеме

В литературе, посвященной шаблонам проектирования, ключевую позицию занимает книга "Приемы объектно-ориентированного проектирования. Паттерны проектирования", авторы Эрих Гамма, Ричард Хелм, Ральф Джонсон и Джон Влиссидес [1]. В ней описывается 23 паттерна проектирования для объектно-ориентированного программирования. Авторы излагают, что такое паттерны проектирования и как с их помощью можно разрабатывать программы. Практическое применение паттернов проектирования демонстрируется на примерах. Каталог занимает большую часть книги и состоит из трех категорий паттернов: порождающие паттерны,

структурные паттерны и паттерны поведения. Описание каждого паттерна разбито на разделы: назначение, мотивация, применимость, структура, участники, отношения, результаты, реализация, пример кода, известные применения и родственные паттерны. Хотя книга позволяет глубоко изучить паттерны для решения задач проектирования, но не содержит единого алгоритма поиска подходящего шаблона.

Еще один труд, посвященный паттернам, - книга "Применение UML 2.0 и шаблонов проектирования", автор Крэг Ларман [2]. Книга описывает основные принципы, современные приемы объектно-ориентированного анализа и проектирования, в нее входят сведения о шаблонах проектирования, архитектурном анализе и многих других вопросах, которые рассматриваются в рамках процесса проектирования UP. На протяжении всей книги рассматривается один реальный пример, в рамках которого описываются возможности применение шаблонов проектирования, но также не приводится алгоритма выбора паттерна.

Подход по адаптации теории решения изобретательских задач для области программирования приведен в статье "Опыт применения методов ТРИЗ для повышение эффективности разработки ПО", авторы Одинцов И.О., Рубин М.С. [3] В статье обосновывается возможность применения методов ТРИЗ для повышения эффективности разработки ПО. Такая возможность появилась благодаря выполненной адаптации классического ТРИЗ для предметной области программирования. Предлагается краткое введение в основные методы и принципы ТРИЗ с учетом программистской терминологии.

Новая система стандартов ТРИЗ для нетехнических систем, которая может быть применена к области программирования, описана в статье " О новой системе стандартов на решение изобретательских задач", автор Рубин М.С. [4] В этой работе приводятся недостатки системы стандартов - 76 и новая система с общесистемными стандартами, применимыми не только для технических систем.

1.4 Основные описания

Атрибут - характеристика материального объекта, отражающая его взаимодействие с другими материальными объектами.

Вещество - объект, имеющий массу покоя.

Вещественно-полевая модель (вепольная модель, веполь) - символическая модель задачи или ее решения, сформулированная в виде взаимодействий между веществами и полями.

Надсистема - система, которая содержит рассматриваемую техническую систему в виде составной части.

Параметр - измеримое значение атрибута.

Подсистема - материальный объект, составляющий часть технической системы.

Поле - объект, не имеющий массы покоя, переносящий взаимодействие между веществами.

Противоречие (техническое) - ситуация, при которой попытка улучшить один параметр системы ведет к недопустимому ухудшению другого параметра.

Система стандартов ТРИЗ - систематизированная совокупность стандартов ТРИЗ.

Стандарты ТРИЗ - правила, указывающие, как должна быть преобразована исходная система для решения типовых задач, которые также сформулированы большей частью в виде вепольных моделей.

Теория решения изобретательских задач (ТРИЗ) - прикладная научная дисциплина, изучающая направления развития и методы совершенствования технических систем, основанные на законах развития технических систем. Предмет ТРИЗ – технические системы и законы развития технических систем.

Техническая система - система, которая предназначена для выполнения некоторой функции.

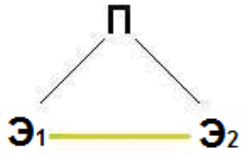
Шаблон (паттерн) проектирования – это архитектурная конструкция, представляющая собой оптимальное решение некоторой проблемы проектирования в рамках конкретных задач, также включающее в себя рекомендации по применению этого решения в различных ситуациях.

Глава 2. Адаптация стандартов ТРИЗ и паттернов проектирования. Создание единого алгоритма

2.1 Адаптация стандартов ТРИЗ к области программирования

Для того, чтобы адаптировать стандарты ТРИЗ для области программирования, в первую очередь нужно переопределить базовые составляющие ТРИЗ и ключевой особенностью здесь является то, что в программировании приходится иметь дело с нематериальными объектами [3].

В связи с тем, что разработчики имеют дело не с веществами, а с абстрактными элементами, минимальную модель системы в программировании можно рассматривать в виде элеполя (двух элементов, связанных тем или иным полем взаимодействия).



Элемент в модели - это визуальное представление абстракции элемента предметной области. Поле - визуальное представление связи между абстракциями элементов предметной области, может представлять собой, например, некоторую функцию или отношение.

Примером элеполя может служить модель "сущность-связь", где элементами являются сущности анализируемой предметной области, а полем - связь между сущностями. Еще один пример - диаграмма классов языка UML, где элементами являются классы, а полями - взаимосвязи между ними.

Как и в классической ТРИЗ, так и в адаптации к области программирования существуют, как минимум, четыре сложности при моделировании с помощью веполей.

1) не существуют системы, которые являются только полем или только веществом или элементом;

2) не всегда понятно, на каком уровне детализации (подсистемы) необходимо создавать веполь или элеполь;

3) не всегда понятно, на каком системном (надсистемном) уровне необходимо строить веполь;

4) в конкретном взаимодействии бывает не одно, а сразу два и больше взаимодействий. Какое из взаимодействий нужно выбрать?

Рекомендуется каждый раз эти неопределенности решать, исходя из конкретных задач моделирования ситуации[3].

Примеры источников полей в программировании:

- механизм взаимодействия между элементами (обменом "сообщениями"). Например, запрос от клиента на выполнение действия объектом (вызов метода объекта);
- отношения между элементами. Например, наследование, ассоциация, агрегация классов.

После переопределения основных понятий, можно адаптировать систему стандартов ТРИЗ к области программирования на основе новой системы стандартов из работы [4]. Подход в новой системе стандартов основан на выделении общесистемных стандартов, применение которых возможно во всех системах, а не только в технических.

В работе вводятся несколько общих правил для элепольных структур:

- элементы могут взаимодействовать только через поля;
- в элеполе поле действует на оба элемента, входящих в элеполь;
- в измерительном элеполе поле преобразуется в другое поле при взаимодействии с элементом;
- объединяться в единое целое могут не только элементы (Э1,Э2), но и элементно-полевые структуры (Э1,П1). Такие структуры носят двойственный характер и рассматриваются либо в качестве вещества, либо в качестве поля в зависимости от рассматриваемой ситуации и решаемой задачи.

Новая система стандартов состоит из двух основных разделов:

- моделей создания и развития элеполей с описанием способов реализации и повышения эффективности этих элепольных структур (стандарты);

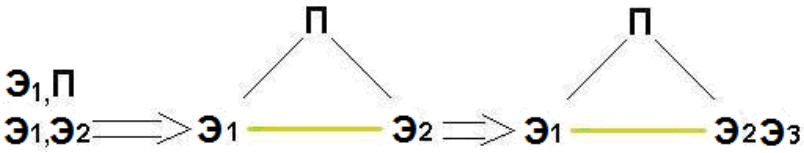
- линии развития систем (универсальное развитие стандарта).

Краткое описание новой системы стандартов:

1. Синтез элеполей

1а Создание элепольной структуры (новой системы)

Если дан объект, плохо поддающийся нужным изменениям, и условия не содержат ограничений на введение элементов и полей, задачу решают синтезом элеполя, вводя недостающие элементы.

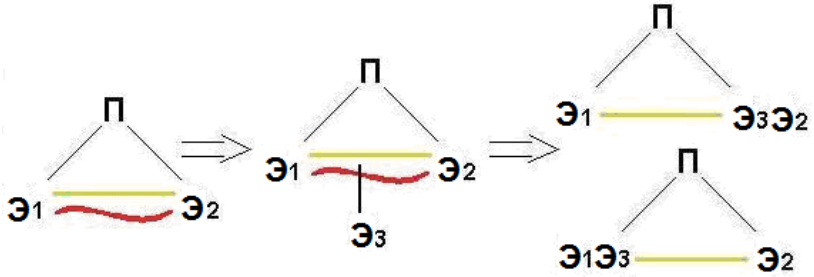


Рекомендуется применить линии введения элементов и полей.

1б Устранение вредных связей в элеполе

1б-1 Устранение вредных связей дополнением элементов

Если между двумя элементами в элеполе возникают сопряженные - полезное и вредное - действия, задачу решают введением постороннего третьего элемента



Элемент Э3 либо нейтрализует, либо оттягивает на себя плохое взаимодействие.

Элемент Э3 в элеполь можно вводить различными способами:

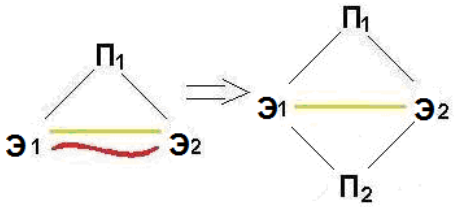
- в виде добавки к Э1 или Э2;

- использовать в качестве Э3 видоизменения Э1 и/или Э2;

- линию введения элементов.

1б-2 Устранение вредных связей дополнением полей

Если между двумя элементами в элеполе возникают сопряженные - полезное и вредное - действия, задачу решают переходом к двойному элеполю, в котором полезное действие остается за полем П1, а нейтрализацию вредного действия (или превращение вредного действия во второе полезное действие) осуществляет П2.

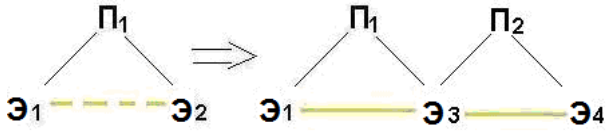


Рекомендуется применять линию введения полей.

2. Развитие элепольных структур

2а Построение цепного элеполя

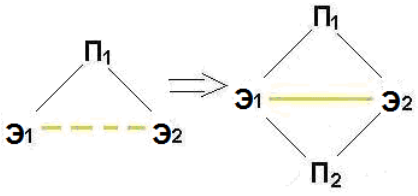
Если нужно повысить эффективность элепольной системы, задачу решают превращением одной из частей элеполя в независимо управляемый элеполь и образованием цепного элеполя:



(Э3 или Э4 в свою очередь может быть развернут в элеполь).

2б Построение двойного элеполя

Если дан плохо управляемый элеполь и нужно повысить его эффективность, причем замена элементов этого элеполя недопустима или нецелесообразна, задача решается постройкой двойного элеполя путем введения второго поля, хорошо поддающегося управлению:

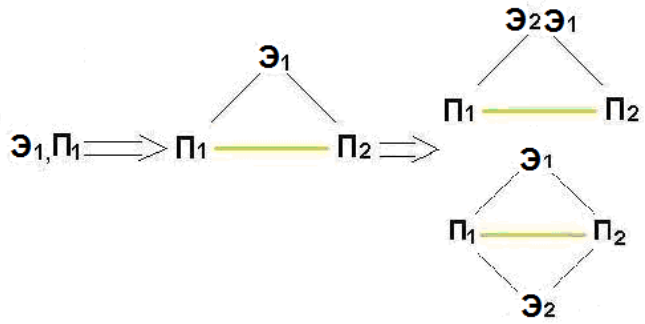


Рекомендовано использовать: линию введения полей, линию введения элементов, структуризацию, динамизацию, согласования.

3. Синтез и повышение эффективности измерительных систем.

Если дана задача на обнаружение или измерение, целесообразно в начале перейти к рекомендациям линии развития измерительных систем и постараться сделать так, чтобы отпала необходимость в измерении.

Если неэлепольная система плохо поддается обнаружению или измерению, задачу решают, достраивая простой, комплексный (Э1-Э2) или двойной элеполь с полем на выходе:



Внешняя среда может быть использована для получения нужной добавки. Для развития измерительных веполей рекомендуется применять линии развития:

- линия развития измерительных систем;
- линии перехода в надсистему и к подсистемам;
- линии введения элементов и полей.

Линии развития:

1. Переход в надсистему и к подсистемам (на микроуровень)

- На любом этапе внутреннего развития система может быть объединена с другими системами в надсистему с новыми качествами:
  - образование бисистем или полисистем;
  - развитием связей внутри бисистем и полисистем;
  - увеличения различий между элементами системы: разные характеристики, разные элементы, противоположные элементы;
  - свертывание би- и полисистем в моносистему с возможным повторением цикла образования полисистем;
  - часть системы наделяется одним свойством, а другая часть или система в целом наделяется противоположным свойством.

- На любом этапе внутреннего развития эффективность системы может быть повышена переходом к развитию подсистемы (на микроуровень), в частности, заменой системы элементом.

Примеры:

1) однотипные данные объединяются в массив, упрощается их применение (доступ к нужным данным осуществляется по индексу);

2) множество браузеров, открытых для отображения множества страниц, свертываются в один, позволяющий хранить страницы во вкладках;

3) в целях уменьшения зависимостей между подсистемами сложной системы, подсистема заменяется элементом, который предоставляет унифицированный интерфейс ко всем возможностям классов подсистемы.

2. Линия развития измерительных систем

- Если дана задача на обнаружение или измерение, целесообразно так изменить систему, чтобы вообще отпала необходимость в решении этой задачи;
- Если это не удается, то целесообразно заменить непосредственные операции над объектом операциями над его копией или снимком;
- Если это не удается, то целесообразно перевести ее в задачу на последовательное обнаружение изменений;
- Эффективность созданной измерительной системы может быть повышена за счет согласования ритмики и использования резонанса, путем перехода к бисистеме и полисистеме.

Примеры:

1) если нужно обнаружить изменения в состоянии некоторого объекта, объект наделяется способностью самостоятельно отсылать уведомление, если меняется его состояние;

2) если по какой-то причине это невозможно, то состояние объекта запрашивается после каждого обращения к нему.

3. Линия введения веществ

- Вместо элемента использовать "пустоту", вместо действия - бездействие;
- Если нужно ввести большое количество элемента, а это запрещено условиями задачи или недопустимо по условиям работы системы, в качестве элемента используют большое количество "пустоты";
- Вместо элемента использовать поле;
- Вместо внутренней добавки использовать наружную добавку;
- Вводить особо активную добавку в очень маленьких дозах;
- Вводить добавку на время;
- Вместо объекта используют его копию (модель), в которую допустимо введение добавки;
- Добавку получают из внешней среды изменением ее в целом или по частям;
- Введенный в систему элемент - после того, как он сработал, - должен исчезнуть или стать неотличимым от элемента, ранее бывшего в системе или во внешней среде.

Примеры:

1) для облегчения понимания кода программы используют комментарии, которые воспринимаются компилятором как "пустота";

2) при пересылке данных в виде блоков, состоящих из байтов, принимающая сторона должна удостовериться, что никакая часть из них не пропала. Для этого вместе с блоками должна передаваться проверочная информация, например, контрольная сумма. Она добавляется в качестве наружной добавки;

3) на этапе тестирования и отладки бывает полезно выводить текущее состояние программы с помощью расположенных в критических точках программы операторов вывода. Затем эти добавки удаляют, т.к. они не должны присутствовать на следующих этапах жизненного цикла.

4. Линия введения полей

- Если в элепольную систему нужно ввести поле, то следует прежде всего использовать уже имеющиеся поля, носителями которых являются входящие в систему элементы;
- При ограничениях на использование полей использовать поля, имеющиеся во внешней среде;
- Если имеются ограничения на введение в систему поля, то следует использовать поля, носителями или источниками которых могут "по совместительству" стать элементы, имеющиеся в системе или во внешней среде.

Пример:

В системе есть абстрактный класс и его подкласс, вводится новый класс. Нужно ввести поле, чтобы обеспечить идентичность интерфейсов подкласса и нового класса. Используем поле, носителем которого является абстрактный класс: наследуем от него этот новый класс.

Линия дробления и динамизации

1. Выделить отдельный элемент, который рассматривается как целое.

2. Разделить элемент на две части (би-элемент) и соединить их между собой полем взаимодействия.

3. Сделать это поле взаимодействия более гибким, динамичным, управляемым, адаптирующимся к ситуации.

4. Разделить элемент не на две, а больше частей (поли-элемент) и соединить их между собой полями взаимодействия.

5. Сделать эти поля взаимодействия более гибкими, динамичными, управляемыми, адаптирующимися к ситуации.

6. Раздробить поли-элемент с динамичными полями взаимодействия до степени возникновения принципиально нового элемента.

7. Новое образование рассмотреть как самостоятельный элемент и изменить его по алгоритму с пункта 1.

Линии согласования-рассогласования и структуризации

На любом этапе развития эффективность функционирования системы может быть повышена за счет согласования входящих в систему элементов и связей между ними.

Пример: согласование скорости работы программы или передачи (обновления) данных со скоростью ее восприятия принимающей стороной (например, человеком). Согласование состояния пользовательского интерфейса с выполняемыми пользователем действиями.

Рассогласование - это обратная сторона согласования. Если необходимо, например, защитить информацию, сделать недоступным изменения, то необходимо максимально рассогласовать возможные потоки информации. При передаче данных в том или ином виде происходит структуризация информация на разных уровнях.

2.2 Перевод паттернов на язык элементов

Паттерны используются для решения задач проектирования в области объектно-ориентированного программирования, следовательно, абстракции, которыми они оперируют - это объекты и классы.

Предполагается, что паттерны проектирования могут помогать в решении задач более высокого архитектурного уровня абстракции, где элементами являются классы и объекты, объединенные в новые структуры - конгломераты. Примером может служить клиент-серверная архитектура.

Понятие объект в этом разделе используется как синоним к понятию элемент. Наследование предполагает, что элемент и его наследники имеют одинаковые свойства и функциональность. Запрос - обращение к элементу с целью выполнения им некоторых действий. Интерфейс отражает множество всех запросов, на которые может отвечать элемент.

Итак, попытаемся перевести паттерны из [1] на язык более высокого уровня абстракции, чем язык классов и объектов, не выходя за пределы

описанной в книге классификации, и параллельно выделяя предназначения конкретных паттернов.

Порождающие паттерны

- **Фабричный метод**

Задает абстрактный способ создания порождающим элементом абстрактного объекта. Порождающий элемент предает ответственность за создание конкретного объекта связанному с ним элементу-потомку, наследующему его структуру и переопределяющему конкретный способ создания.

Признаки применения:

- Порождающему элементу заранее неизвестно, какие конкретные объекты ему нужно создавать, поскольку предполагается много различных вариантов функционирования системы;
- Порождающий элемент определен так, чтобы объекты, которые он создает, устанавливались только в его наследниках;
- Порождающий элемент передает свои обязанности одному из нескольких вспомогательных наследников, т.е. функциональность логически разделена между ним и вспомогательными элементами, каждый из которых выполняет свою конкретную функцию. Паттерн предоставляет возможность легко заменять эти вспомогательные элементы.

- **Абстрактная фабрика**

Предоставляет способ для создания группы объектов. Элемент (абстрактная фабрика) содержит абстрактный способ для создания семейства абстрактных объектов-продуктов. Конкретные способы создания конкретных семейств определяются в элементах-наследниках (конкретных фабриках). В процессе работы создается только один наследник, который создает конкретное семейство объектов-продуктов.

Признаки применения:

- Система не должна зависеть от того, как создаются объекты.

- Конкретный вариант требуемого поведения системы дают не отдельные объекты, а четко выраженное семейство связанных объектов. Объекты одного семейства должны использоваться вместе.
- Для функционирования системы необходимо одно из таких семейств взаимосвязанных объектов.

- **Одиночка**

Гарантирует, что у элемента будет не более одного созданного экземпляра, предоставляет к нему доступ.

Признаки применения:

Должен быть ровно один легко доступный экземпляр некоторого элемента.

- **Прототип**

Определяет, задает виды создаваемых объектов с помощью некоторого элемента-прототипа и создает новые объекты путем его копирования (клонирования).

Признаки применения:

- Порождаемые объекты определяются в процессе функционирования системы;
- Для того чтобы избежать построения иерархий конкретных фабрик, параллельных иерархии объектов-продуктов;
- Создаваемые объекты могут находиться в одном из небольшого числа различных состояний, так что может оказаться целесообразней устанавливать соответствующее число прототипов и клонировать их.

- **Строитель**

Строитель – паттерн, конструирующий объекты.

Если нужно создавать различные сложные объекты, при этом процесс создания их всех протекает очень схожими этапами, так что можно выделить общий единый алгоритм, включающий выполнение всех возможных действий. При этом конструируемые объекты в общем случае не схожи

между собой, т.е. их невозможно обобщить все в единую иерархию, объединить общие свойства и т.д.

Тогда, чтобы не дублировать действия при создании, этот процесс можно объединить в единый алгоритм, за выполнение которого отвечает некоторый элемент, способный создать любой из сложных объектов.

Так как объекты совершенно разные, необходимо чтобы каждый из них имел свою конкретную структуру. Для этого следует выделить еще один элемент, отвечающий непосредственно за создание отдельных частей. Эта сущность будет своя для каждого сложного объекта, который может быть построен.

Признаки применения:

- Алгоритм создания сложного объекта не должен зависеть от того, из каких частей он состоит, и как они стыкуются между собой.
- Процесс должен позволять создавать различные виды конечного объекта.

Структурные паттерны. Структурные паттерны помогают найти способ оптимального получения более крупных структур и образований из уже имеющихся элементов.

- **Адаптер**

Адаптер обеспечивает совместную работу элементов с несовместимыми интерфейсами, которая без него была бы невозможна. Адаптер класса представляет собой элемент - потомок элементов с несовместимыми интерфейсами, наследующий их функциональности. Адаптер объекта наследует требуемый интерфейс и содержит в себе экземпляр элемента с адаптируемым интерфейсом, тем самым имея доступ к его функциональности.

Признаки применения:

- Необходимо использовать существующий элемент, но его интерфейс не соответствует потребностям.

- Необходимо создать повторно используемый элемент, который должен взаимодействовать с заранее неизвестными или не связанными с ним объектами, имеющими несовместимые интерфейсы.

- **Декоратор**

Декоратор - паттерн, структурирующий элементы, расширяя их функциональность.

Динамически расширяет функциональность элемента, добавляет ему новые обязанности.

Признаки применения:

- Для динамического добавления новых возможностей элементам.
- Для реализации возможностей, которые нужны не всем элементам и не всегда, которые потом можно легко исключить.
- Когда расширение функциональности путем порождения элементов-потомков с новой функциональностью по каким-то причинам неудобно или невозможно.

- **Заместитель**

Заместитель - паттерн, контролирующий доступ к элементам, предоставляя более оптимальное их взаимодействие.

Разумно управлять доступом к элементу, поскольку тогда можно отложить расходы на его создание до момента, когда элемент действительно понадобится. Таким образом, выявляются элементы, функционирование которых проходит не совсем оптимально, и вводятся объекты-заместители, которые, дублируя внешний вид и поведение «проблемных» элементов, переадресуют им запросы лишь тогда, когда это действительно необходимо, либо после некоторых оптимизационных действий.

Признаки применения:

- Когда требуется удаленный функционал

Удаленный заместитель предоставляет локального представителя вместо целевого объекта, находящегося в другом (адресном) пространстве.

- Когда нужен виртуальный заместитель

Виртуальный заместитель создает «тяжелые» элементы по требованию.

- Когда нужно контролировать доступ к исходному элементу

Защищающий заместитель контролирует доступ к исходному элементу.

- **Компоновщик**

Компоновщик - паттерн, структурирующий элементы таким образом, что появляется возможность одинаковым образом обращаться с каждым из них. Позволяет единообразно трактовать индивидуальные и составные элементы.

Признаки применения:

- Нужно представить иерархию различных элементов так, чтобы с каждой ее веткой можно было работать одинаково, как с одной и той же сущностью.
- Чтобы можно было единообразно трактовать составные и индивидуальные элементы.

- **Мост (только для ООП)**

Главное назначение - отделить абстракцию от ее реализации так, чтобы то и другое можно было изменять независимо. Польза данного разделения появляется в случае, когда для некоторой абстракции возможно несколько реализаций.

Признаки применения:

- Нужно избежать постоянной привязки абстракции к реализации.
- Когда конкретную реализацию необходимо выбирать во время функционирования системы.
- Нужно разделить одну и ту же реализацию между несколькими элементами.

- **Приспособленец**

Приспособленец - паттерн, структурирующий элементы таким образом, что из них создается всего лишь ограниченный необходимый набор экземпляров вместо всего большого множества.

У элементов из множества выделяется фиксированное число их состояний, которые они принимают в процессе своей деятельности, на каждое такое состояние создается по специальному объекту и далее в процессе функционирования в программе создается и фигурирует только это ограниченное количество экземпляров.

Эффективность паттерна приспособленец во многом зависит от того, как и где он используется. Следует применять этот паттерн, когда выполнены все нижеперечисленные условия:

- Используется большое число элементов.
- Большую часть состояния элементов можно вынести вовне системы.
- Система не зависит от идентичности элемента.

- **Фасад**

Фасад - паттерн, структурирующий элементы, предоставляя ко всем ним доступ через единый элемент.

Предоставляет единый, унифицированный интерфейс ко всей некоторой подсистеме вместо набора отдельных и многочисленных интерфейсов.

Признаки применения:

- Нужно предоставить простой интерфейс к сложной подсистеме
- Между элементами, использующими подсистему, и элементами системы существует много зависимостей. Нужно изолировать подсистему.
- Нужно разложить систему на отдельные слои.

Паттерны поведения. Паттерны поведения связаны с алгоритмами, способами взаимодействия, связями и распределением обязанностей между элементами.

- **Интерпретатор**

В ряде случаев система на разных этапах использует одни и те же алгоритмы, строящиеся из более простых, элементарных неизменяющихся подалгоритмов.

Паттерн интерпретатор предлагает составлять лишь некоторую схему компоновки новых алгоритмов этими элементарными составляющими и далее передавать ее в некоторый элемент, который собирает полноценную архитектуру данного алгоритма.

- **Шаблонный метод**

Шаблонный метод - паттерн поведения объектов, определяющий функциональность конкретных методов в рамках абстрактных сущностей.

Шаблонный метод определяет основу алгоритма в рамках абстрактных элементов и методов, а элементам-потомкам позволяет переопределять отдельные шаги этого алгоритма (или все сразу), не изменяя, таким образом, его структуру и последовательность в целом.

Признаки применения

- Чтобы однократно использовать инвариантные части алгоритма.
- Когда нужно вычленить и локализовать в одном элементе поведение.

- **Итератор**

Итератор – паттерн поведения объектов, предоставляющий последовательный доступ ко всем элементам составного объекта, не раскрывая его внутреннего представления.

Признаки применения:

- Для доступа к содержимому агрегированных элементов без раскрытия их внутреннего представления.
- Для поддержки нескольких видов активных обходов одного и того же агрегированного элемента.

- **Команда**

Представляет запрос на выполнение конкретного алгоритма в виде элемента, позволяя тем самым задавать параметры для обработки соответствующих запросов, ставить запросы в очередь или протоколировать их, а также поддерживать отмену операций.

Признаки применения:

- Нужно параметризовать объекты выполняемым действием.

- Определять, ставить в очередь и выполнять запросы в разное время.
- Поддержать отмену операций.
- Поддержать протоколирование изменений.
- **Наблюдатель**

Наблюдатель - паттерн поведения объектов, устанавливающий систему оповещения объектами своих соседей в процессе их деятельности.

Удобно иметь такую структуру, в которой каждый элемент, если он заинтересован в каких-либо событиях системы, мог бы самостоятельно «подписаться» на эти изменения независимо от других заинтересованных участникам и, таким образом, получая уведомления об этих событиях, выполнять требуемые действия.

Наблюдатель определяет зависимость типа «один ко многим» между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом и автоматически обновляются.

Признаки применения:

- Когда у элементов существуют параметры, один из которых зависит от другого.
- Когда при модификации одного объекта требуется изменить и другие, но неизвестно, сколько именно объектов нужно изменить.

Когда один объект должен оповещать других, не владея информацией об уведомляемых объектах.

- **Посредник**

Посредник - паттерн поведения элементов, предоставляющий единый центр взаимодействия определенной группы элементов, которые должны быть взаимосвязаны друг с другом. Определяет элемент, содержащий сведения о способах взаимодействия множества элементов.

Посредник обеспечивает слабую связанность системы, избавляя объекты от необходимости явно ссылаться друг на друга, и позволяя тем самым независимо изменять взаимодействия между ними.

Признаки применения:

- Имеются элементы, связи между которыми сложны и четко определены.
- Нельзя повторно и независимо использовать элемент (например, в другом контексте), поскольку он обменивается информацией со многими другими элементами.

- **Посетитель**

Посетитель - паттерн поведения объектов, задающий стратегии обхода.

Описывает операцию, которая должна быть выполнена над каждым элементом из некоторой произвольной структуры.

Признаки применения:

- В структуре присутствуют объекты с различными интерфейсами и нужно выполнять операции, не зависящие от интерфейсов.
- Над элементами, входящими в состав структуры, надо выполнять разнообразные, не связанные между собой.
- Классы, устанавливающие структуру объектов, изменяются редко, но новые операции над этой структурой добавляются часто.

- **Состояние**

Состояние - паттерн поведения объектов, задающий разную функциональность в зависимости от состояния элемента.

Позволяет элементу варьировать свое поведение в зависимости от внутреннего состояния. Поскольку поведение может меняться совершенно произвольно без каких-либо ограничений, извне создается впечатление, что изменился элемент.

Признаки применения

- Когда поведение элемента зависит от его состояния и при этом должно изменяться во время функционирования.
- Когда существует множество условий и выбор ветви зависит от состояния. Паттерн позволяет трактовать состояние элемента как самостоятельный элемент, который может изменяться независимо от других.

- **Стратегия**

Если в системе фигурируют различные алгоритмы, которые часто могут использоваться повторно в других частях, удобно поместить каждый из них в отдельный элемент, параметризовать и запускать их там, где понадобится. Стратегия определяет семейство алгоритмов и делает их взаимозаменяемыми.

Признаки применения:

- Имеется много родственных элементов, отличающихся только поведением.
- Нужно иметь несколько разных вариантов алгоритма.
- В элементе определено много поведений.

- **Хранитель**

Хранитель - паттерн поведения элементов, сохраняющий состояния.

Фиксирует и выносит за пределы элемента его внутреннее состояние так, чтобы позднее можно было восстановить в нем элемент.

Признаки применения:

Необходимо сохранить мгновенный снимок состояния элемента (или его части), чтобы впоследствии элемент можно было восстановить в том же состоянии.

- **Цепочка обязанностей**

Цепочка обязанностей - паттерн поведения, выстраивающий элементы составных частей системы связанными между собой по цепочке, для передачи запроса на обработку от более низких, детализированных слоев системы к более высоким глобальным. Позволяет избежать привязки отправителя запроса к его получателю, связывает элементы-получатели в цепочку и передает запрос вдоль этой цепочки, пока его один из получателей его не обработает.

Признаки применения:

- Есть более одного элемента, способного обработать запрос, при этом настоящий обработчик заранее неизвестен и должен быть найден в соответствии с какой-либо логикой.
- Нужно отправить запрос одному из нескольких элементов, не указывая явно, какому именно.

Таким образом паттерны проектирования удалось перевести на общесистемный язык, а значит их возможно использовать в едином алгоритме решения задач программирования на ряду со стандартами ТРИЗ.

2.3 Единый алгоритм решения задач из области программирования

Данный алгоритм АИСТ-2010-П основан на алгоритме применения стандартов на решение изобретательских задач - 2010 [5], включает в себя расширение в область программирования для выбора подходящего паттерна проектирования.

В ходе анализа структуры шаблонов было определено, что их визуальное представление не укладывается в стандартные элепольные модели, представляя собой графовые структуры. Поэтому в единый алгоритм будет включены лишь наиболее общие признаки выбора подходящего паттерна, пользователю предлагается самостоятельно развивать модель задачи на основе структурных моделей паттернов.

1 **Формулировка задачи**. Первый этап алгоритма - анализ проблемной ситуации, в результате которого задача должна быть сформулирована в терминах полей и элементов с параметрами.

1.1 Описание проблемной ситуации.

1.2 На основе описания выделяются объекты-претенденты на участие в построении элеполя.

1.3 Параметризация объектов. Для каждого объекта выделяются параметры в зависимости от аспекта рассмотрения ситуации.

Следует учитывать характеристики параметра:

- Параметр характеризует состояние объекта.
- Изменить значение параметра можно только воздействуя на объект.
- Время является параметром только для процессов или операций.
- Параметр можно измерить тем или иным способом, включая экспертные оценки.
- Для одного и того же параметра существуют не менее двух объектов, характеризующихся этим параметром, параметр не может быть уникальным только для одной системы.
- Параметр можно увеличивать, уменьшать, стабилизировать, управлять.
- Параметры объекта могут быть взаимосвязанными между собой.
- Взаимная связь между параметрами объекта определяется свойствами этого объекта.
- Объект может характеризоваться разными параметрами в зависимости от аспекта его рассмотрения.
- Параметры объекта могут быть связаны причинно-следственными цепочками и создавать иерархические параметрические структуры[6].

1.4 Создание элепольной структуры. На этом этапе формулируется решаемая задача, из списка объектов-претендентов выделяется один или два элемента и формируется неполный или полный элеполь соответственно. Выбор элементов должен отражать суть проблемной ситуации, то есть определять какие объекты возможно должны измениться для того, чтобы проблема была решена.

2 **Анализ задачи**. На этом этапе проводится анализ задачи на основе алгоритма применения стандартов ТРИЗ. Результатом являются рекомендации по использованию стандартов для решения задачи.

2.1 Если суть задачи – обнаружение или измерение, переход к п. 2.1.1, иначе – п. 2.2

2.1.1 Если дана задача на обнаружение или измерение, целесообразно так изменить систему, чтобы вообще отпала необходимость в решении этой задачи.

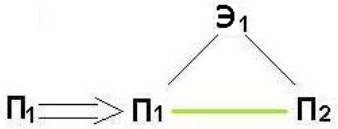
Если это не удается, то целесообразно заменить непосредственные операции над объектом операциями над его копией.

Если это не удается, то целесообразно перевести ее в задачу на последовательное обнаружение изменений.

Если поле из п. 1.4 единственное и плохо поддается измерению, рекомендуется переход к стандартам на создание измерительных систем:

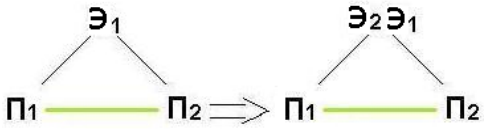
Создание измерительной системы

Если поле П1 плохо поддается измерению и условия задачи не содержат ограничений на введение элементов и полей, то задачу решают синтезом элеполя с полем на выходе.



Элемент Э1 и поле П2 могут быть выбраны из п. 1.2, могут быть введены новый элемент и новое поле.

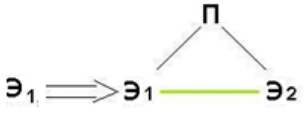
Если есть два поля и элемент и система плохо поддаются обнаружению или измерению, то в Э1 вводится легко обнаружимая добавка Э2.



2.2 Если элемент из п. 1.4 единственный, рекомендуется переход к стандартам на создание элепольных структур п. 2.2.1

2.2.1 Создание элепольной структуры (новой системы)

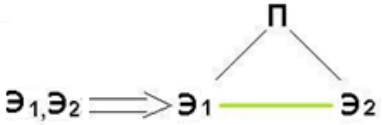
Если дан объект, плохо поддающийся нужным изменениям, и условия не содержат ограничений на введение элементов и полей, задачу решают синтезом элеполя, вводя недостающие элементы.



Элемент Э2 и поле П могут быть выбраны из п. 1.2, могут быть введены новый элемент и новое поле.

Если в п. 1.4 два элемента - переход к п. 2.2.2 или 2.2.3:

2.2.2 Если между элементами нет взаимодействия - переход к стандартам на создание элепольных структур.



Вводится новое поле П. Переход к 2.3 или 2.4

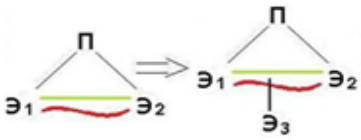
2.2.3 Если между элементами есть взаимодействие, определяется каким полем П оно создается. Переход к 2.3 ли 2.4

2.3 Если между элементами возникает одновременно полезное и вредное взаимодействие, переход к стандартам на устранение вредных связей.

Устранение вредных связей в элеполе

- Устранение вредных связей дополнением элементов.

Если между двумя элементами в элеполе возникают сопряженные - полезное и вредное - действия, задачу решают введением постороннего третьего элемента:

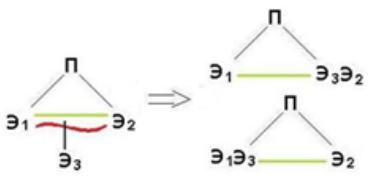


Элемент Э3 либо нейтрализует, либо оттягивает на себя плохое взаимодействие.

Элемент Э3 в элеполь можно вводить различными способами:

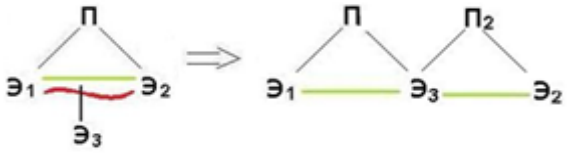
- в виде добавки Э3 к элементу Э1 или Э2

- использовать в качестве добавки Э3 видоизменения Э1 и/или Э2



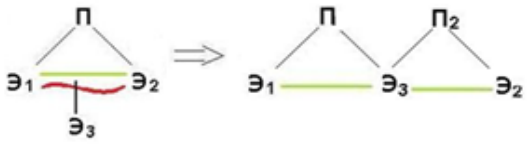
Если введение добавок не разрывает вредную связь:

- Э2 и Э3 образуют независимый элеполь, с которым у Э1 полезное взаимодействие сохраняется, а вредное нет:



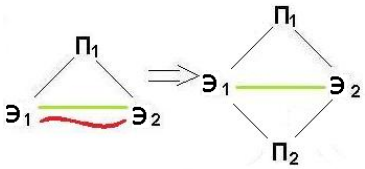
Если элементы невозможно или нецелесообразно изменять:

- в связь между Э1 и Э2 встраиваем новое звено, которое нейтрализует вредное взаимодействие между Э1 и Э2



- Устранение вредных связей дополнением полей.

Если между двумя элементами в элеполе возникают сопряженные - полезное и вредное - действия, задачу решают переходом к двойному элеполю, в котором полезное действие остается за полем П1, а нейтрализацию вредного действия (или превращение вредного действия во второе полезное действие) осуществляет П2.

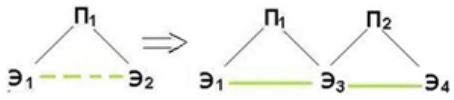


2.4 Если нужно повысить эффективность взаимодействия, определяем параметр системы, который при этом должен измениться (и способ изменения) и параметр одного из элементов, изменение которого к этому приводит. Если ни у одного из элементов нет такого параметра, переход к п. 2.4.1, иначе переход к 2.4.2.

2.4.2 Развитие элепольных структур

- Построение цепного элеполя.

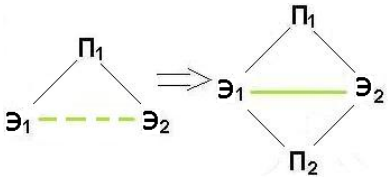
Если нужно повысить эффективность элепольной системы, задачу решают превращением одной из частей элеполя в независимо управляемый (значение параметра меняется) элеполь и образованием цепного элеполя



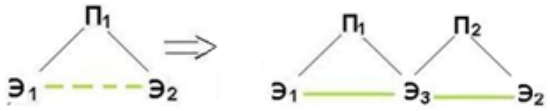
(Э3 или Э4 сами в свою очередь могут быть развернуты в элеполь.)

- Построение двойного элеполя.

Если нужно повысить эффективность элеполя, причем замена элементов этого элеполя нецелесообразна, задача решается постройкой двойного элеполя путем введения второго поля, усиливающего эффективность взаимодействия между элементами (поле воздействует на параметр):



2.4.1 Если изменение параметров элементов не приводит к повышению эффективности, в связи между элементами можно ввести новое звено, увеличивающее эффективность системы:



2.5 Если не удалось подобрать подходящий стандарт, возможно задача сформулирована неверно - переход к 1.4 или задача нестандартная и рекомендуется перейти к решению по АРИЗ-82 или его модификациям.

2.6 Для развития элеполя рекомендуется перейти к линиям развития.

3. Если удалось найти подходящий стандарт - переход к выбору паттерна по ключевым особенностям применения:

3.1 Если нужно устранить вредную связь:

- Если один элемент (создатель) содержит алгоритм создания второго элемента и при этом:
  - Нужно заменять порождаемый элемент без изменения создателя. **Фабричный метод**
  - Нужно заменять семейство порождаемых элементов без изменения создателя. **Абстрактная фабрика**
  - Алгоритм создания элемента не должен зависеть от того, из каких частей и в какой их комбинации состоит порождаемый элемент. **Строитель**
- Интерфейс одного элемента не должен зависеть от интерфейса второго. **Адаптер**
- Нужно создавать элемент/получать к нему доступ только в конкретных случаях. **Заместитель**
- Нужно одинаково обращаться к элементам, не зависимо, являются они простыми и составными. **Компоновщик**
- Нужно отделить абстракцию элемента от реализации. **Мост**
- Нужно снизить зависимость между подсистемами. **Фасад**
- Нужно обеспечить слабую связанность между элементами системы. **Посредник**
- Нужно избежать связи между отправителем запроса и получателем. **Цепочка обязанностей**

3.2 Если нужно повысить эффективность:

- Если один элемент (создатель) содержит алгоритм создания второго элемента и при этом:
  - Нужно переопределить алгоритмы создания элементов в элементах-потомках. **Фабричный метод**
  - Нужно переопределить алгоритмы создания семейств элементов в элементах-потомках. **Абстрактная фабрика**
  - Нужно гарантировать, что элемент создан в единственном экземпляре. **Одиночка**

- Новые элементы создаются путем копирования элемента-прототипа. **Прототип**

- Нужно обеспечить совместное использование элементов с различными интерфейсами. **Адаптер**
- Нужно динамически расширить функциональность элемента, добавить ему новые обязанности на время. **Декоратор**
- Нужно заменить элемент до момента, когда он действительно понадобится. **Заместитель**
- Нужно одинаково обращаться с простыми и составными элементами. **Компоновщик**
- Нужно ограничить набор экземпляров элементов. **Приспособленец**
- Нужно предоставить доступ к элементам подсистемы с помощью одного элемента. **Фасад**
- Нужно переопределить шаги алгоритма в элементах-потомках. **Шаблонный метод**
- Нужно предоставить последовательный доступ ко всем подэлементам составного элемента. **Итератор**
- Нужно представить запрос в виде элемента, ставить запросы в очередь, поддерживать отмену операций. **Команда**
- Элементы должны изменять свое состояние в зависимости от состояния других элементов. **Наблюдатель**
- Нужно объединить все связи между элементами в одном элементе. **Посредник**
- Нужно обойти элементы структуры, выполнив над каждым из них некоторую операцию. **Посетитель**
- Нужно изменять поведение элемента в зависимости от его состояния. **Состояние**
- Нужно определить семейство взаимозаменяемых алгоритмов в виде элементов. **Стратегия**

• Нужно сохранить состояние элемента, восстановить элемент в нужном состоянии. **Хранитель**

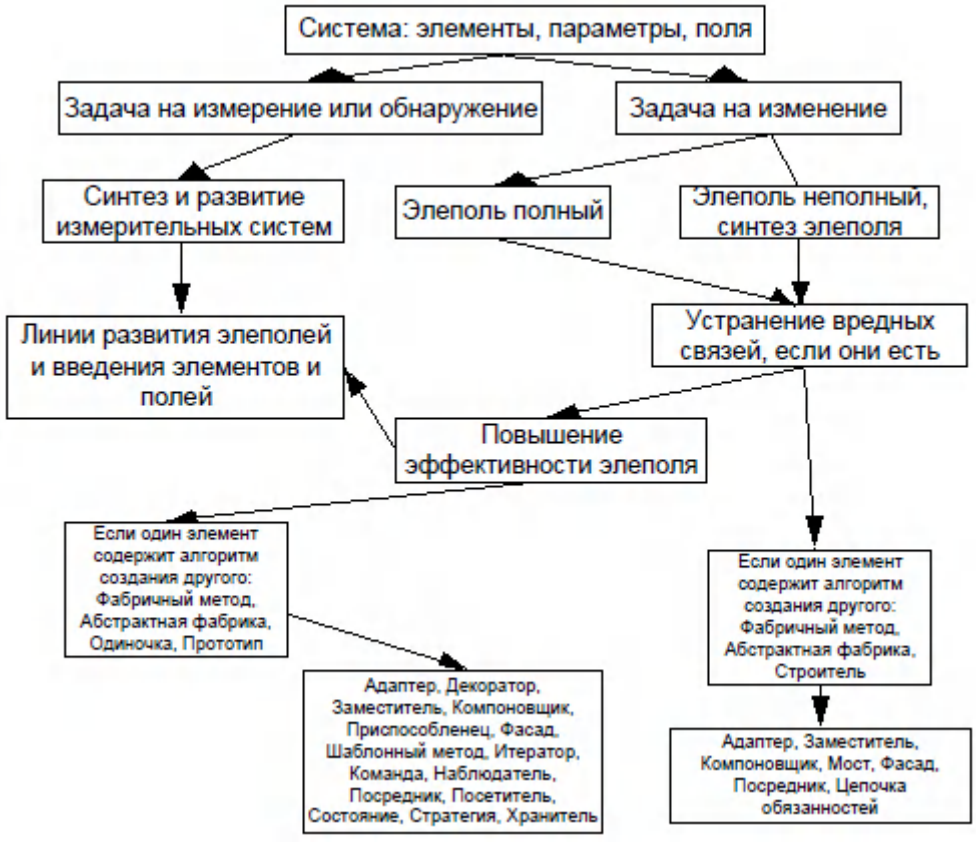
3.3 Если удалось выбрать подходящий вариант, переход к описанию подходящего паттерна.

3.4 Если паттерн по описанию все же не соответствует желаемому результату, переход к группе родственных паттернов.

3.5 Если в группе не удалось подобрать подходящий вариант, возможно задача не должна решаться с помощью паттернов.

3.6 Для подходящего паттерна предоставляются рекомендации по использованию совместно с другими паттернами.

Графическое представление алгоритма:



2.4 Примеры решения задач по алгоритму

Задача 1

1 **Формулировка задачи**.

1.1 Описание проблемной ситуации.

Рассмотрим редактор документов, который допускает встраивание в документ графических объектов. Затраты на создание некоторых таких

объектов, например больших растровых изображений, могут быть весьма значительны. Но документ должен открываться быстро, поэтому следует избегать создания всех «тяжелых» объектов на стадии открытия (да и вообще это излишне, поскольку не все они будут видны одновременно). Разумно создавать «тяжелые» объекты по требованию. Это означает «когда изображение становится видимым». Как это сделать?

1.2 На основе описания выделяются объекты-претенденты на участие в построении элеполя.

Редактор, документ, графический объект.

1.3 Параметризация объектов. Для каждого объекта выделяются параметры в зависимости от аспекта рассмотрения ситуации.

Редактор: размер кода.

Документ: количество встроенных графических объектов.

Графический объект: ширина, высота изображения, объем памяти, занимаемый изображением.

1.4 Создание элепольной структуры.

Э1 - редактор, Э2 - графический объект.

2 **Анализ задачи**

2.1 Если суть задачи – обнаружение или измерение, переход к п. 2.1.1, иначе – п. 2.2

2.2 Если в п. 1.4 два элемента переход к п. 2.2.2 или 2.2.3:

2.2.3 Если между элементами есть взаимодействие, определяется каким полем П оно создается. Переход к 2.3 ли 2.4

Поле П - запросы к графическому объекту.

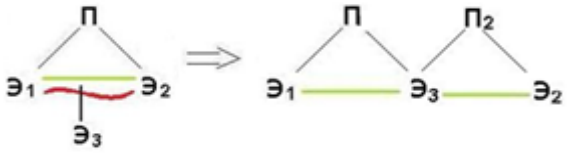
2.3 Если между элементами возникает одновременно полезное и вредное взаимодействие, переход к стандартам на устранение вредных связей.

Полезное взаимодействие обеспечивает возможность отображения и работы с изображением, вредное - создание изображения, пока оно не

появляется в видимой части документа, снижает скорость работы системы.

Устранение вредных связей в элеполе

- Устранение вредных связей дополнением элементов Э2 и Э3 образуют независимый элеполь, с которым у Э1 полезное взаимодействие сохраняется, а вредное нет:



Вводится новый объект Э3, который ведет себя так же и отображает настоящее изображение, на которое хранит ссылку, только в необходимых случаях. Э3 хранит размер изображения, может отвечать на запросы о своем размере, не создавая его.

3. Если удалось найти подходящий стандарт переход к выбору паттерна по ключевым особенностям применения:

3.1 Если нужно устранить вредную связь:

- Нужно создавать элемент/получать к нему доступ только в конкретных случаях. **Заместитель**

3.3 Если удалось выбрать подходящий вариант, переход к описанию подходящего паттерна.

Заместитель - паттерн, контролирующий доступ к элементам, предоставляя более оптимальное их взаимодействие.

Разумно управлять доступом к элементу, поскольку тогда можно отложить расходы на его создание до момента, когда элемент действительно понадобится. Таким образом, выявляются элементы, функционирование которых проходит не совсем оптимально, и вводятся объекты-заместители, которые, дублируя внешний вид и поведение «проблемных» элементов, переадресуют им запросы лишь тогда, когда это действительно необходимо, либо после некоторых оптимизационных действий.

Приводятся признаки применения, графическая модель, достоинства и недостатки.

Задача 2

1.1 Описание проблемной ситуации.

Была написана программа, работающая в грид-среде. В процессе использования выяснилось, что скорость работы программы в грид соизмерима со скоростью работы на одном компьютере. Хотя предполагается, что скорость работы в распределенной среде должна быть значительно выше. Как быть?

1.2 На основе описания выделяются объекты-претенденты на участие в построении элеполя.

Программа, распределенная среда (грид).

1.3 Параметризация объектов. Для каждого объекта выделяются параметры в зависимости от аспекта рассмотрения ситуации.

Программа: гранулярность, размер кода, количество пересылок при обмене данными с компьютерами среды.

Грид-среда: количество компьютеров, средняя скорость связи.

1.4 Создание элепольной структуры.

Э1 - программа, Э2 - грид-среда.

2 **Анализ задачи**.

2.1 Если суть задачи – обнаружение или измерение, переход к п. 2.1.1, иначе – п. 2.2

Если в п. 1.4 два элемента переход к п. 2.2.2 или 2.2.3:

2.2.3 Если между элементами есть взаимодействие, определяется каким полем П оно создается. Переход к 2.3 или 2.4.

П - алгоритм распределения задач между компьютерами среды.

2.4 Если нужно повысить эффективность взаимодействия, определяем параметр системы который при этом должен измениться (и способ изменения) и параметр одного из элементов, изменение которого к этому

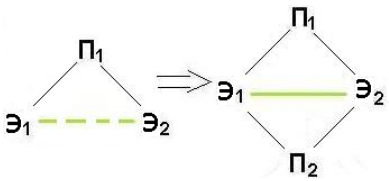
приводит. Если ни у одного из элементов нет такого параметра, переход к п. 2.4.1, иначе переход к 2.4.2.

Параметр системы: скорость работы программы в распределенной среде, повысить.

Параметр элемента Э1: гранулярность.

2.4.2 Развитие элепольных структур

- Построение двойного элеполя.



Вводится поле П2 - алгоритм изменения парамера "гранулярность" в зависимости от состояния среды (средняя скорость связи компьютеров).

Источником поля П2 по совместительству выступает элемент Э1 (встраивается программный модуль, инкапсулирующий семантику обновления).

3. Если удалось найти подходящий стандарт переход к выбору паттерна по ключевым особенностям применения:

3.2 Если нужно повысить эффективность:

- Элементы должны изменять свое состояние в зависимости от состояния других элементов. **Наблюдатель.**

3.3 Если удалось выбрать подходящий вариант, переход к описанию подходящего паттерна.

Наблюдатель - паттерн поведения элементов, устанавливающий систему оповещения объектами своих соседей в процессе их деятельности.

Удобно иметь такую структуру, в которой каждый элемент, если он заинтересован в каких-либо событиях системы, мог бы самостоятельно «подписаться» на эти изменения независимо от других

заинтересованных участникам и, таким образом, получая уведомления об этих событиях, выполнять требуемые действия.

Наблюдатель определяет зависимость между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом и автоматически обновляются.

Приводятся признаки применения, графическая модель, достоинства и недостатки.

3.6 Для подходящего паттерна предоставляются рекомендации по использованию совместно с другими паттернами.

Посредник: новый программный модуль действует как посредник между Э1 и Э2.

Одиночка: может воспользоваться паттерном одиночка, чтобы гарантировать свою уникальность и глобальную доступность.

Задача 3

1.1 Описание проблемной ситуации.

Процесс разработки приложения велся N лет, к текущему варианту постепенно добавлялась новая функциональность. Анализ выявил, что 20% всего кода составляет полезный (функциональный) код, а 80% - неоптимизированный связующий код. Нужно оптимизировать код программы.

1.2 На основе описания выделяются объекты-претенденты на участие в построении элеполя.

Программа, компоненты [1-N] (в них входит функциональный и связующий код).

1.3 Параметризация объектов. Для каждого объекта выделяются параметры в зависимости от аспекта рассмотрения ситуации.

Программа: размер кода, размер полезного кода, размер связующего кода.

Компоненты: размер функционального кода, размер связующего кода.

1.4 Создание элепольной структуры.

Э1 - компонент [i], Э2 - компонент [j], т.е. любая пара компонентов.

2 **Анализ задачи**.

2.1 Если суть задачи – обнаружение или измерение, переход к п. 2.1.1, иначе – п. 2.2.

2.2 Если в п. 1.4 два элемента переход к п. 2.2.2 или 2.2.3:

2.2.3 Если между элементами есть взаимодействие, определяется каким полем П оно создается. Переход к 2.3 ли 2.4.

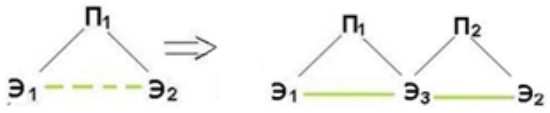
Поле П1 - связи между компонентами.

2.4 Если нужно повысить эффективность взаимодействия, определяем параметр системы который при этом должен измениться (и способ изменения) и параметр одного из элементов, изменение которого к этому приводит. Если ни у одного из элементов нет такого параметра, переход к п. 2.4.1, иначе переход к 2.4.2.

Параметр системы: размер связующего кода, понизить.

У элементов нет подходящего параметра. (Параметры "размер связующего кода" в совокупности по всем компонентам и дает нужный параметр системы, значит нужно найти способ как изменить их самих).

2.4.1 Если изменение параметров элементов не приводит к повышению эффективности, в связи между элементами можно ввести новое звено, увеличивающее эффективность системы:



Вводится новый элемент Э3 - компонент, инкапсулирующий логику взаимодействия между Э1 и Э2, содержит оптимизированный связующий код, который не нужно использовать в Э1 и Э2.

3. Если удалось найти подходящий стандарт переход к выбору паттерна по ключевым особенностям применения:

3.2 Если нужно повысить эффективность:

- Нужно объединить все связи между элементами в одном элементе.

Посредник

3.3 Если удалось выбрать подходящий вариант, переход к описанию подходящего паттерна.

Посредник - паттерн поведения элементов, предоставляющий единый центр взаимодействия определенной группы элементов, которые должны быть взаимосвязаны друг с другом. Определяет элемент, содержащий сведения о способах взаимодействия множества элементов.

Посредник обеспечивает слабую связанность системы, избавляя объекты от необходимости явно ссылаться друг на друга, и позволяя тем самым независимо изменять взаимодействия между ними.

Приводятся признаки применения, графическая модель, достоинства и недостатки.

3.6 Для подходящего паттерна предоставляются рекомендации по использованию совместно с другими паттернами.

Элементы могут обмениваться информацией с посредником посредством паттерна наблюдатель.

Глава 3. Проектирование и реализация системы

3.1 Описание системы и основного алгоритма

Основной алгоритм системы представляет программную реализацию алгоритма подбора *стандартов ТРИЗ на изменение систем* и паттернов проектирования для решения задач из области программирования.

Входными данными является описание проблемной ситуации, на выходе предлагаются рекомендации по использованию паттернов и стандартов ТРИЗ.

Описание шагов алгоритма:

Шаг 1. Пользователь вводит описание проблемной ситуации в терминах предметной области.

Шаг 2. Из условия пользователь выбирает объекты, значимые с точки зрения проблемы.

Шаг 3. Для каждого объекта вводятся параметры, зависящие от аспекта рассмотрения.

Шаг 4. Из списка пользователь выбирает один или два объекта, в терминах которых формулируется задача.

Шаг 5.1. Если выбран один элемент, пользователь вводит второй элемент с параметрами и поле, связывающее элементы.

Шаг 5.2. Если выбраны два элемента, пользователь вводит название поля, которым определяется взаимодействие между элементами.

Шаг 6. Пользователь определяет нужно ли повысить эффективность системы или существует ли между элементами одновременно и полезное, и вредное взаимодействие.

Если указано, что нужно повысить эффективность системы, пользователь вводит параметр системы, который нужно изменить, и определяет действие над этим параметром (увеличить, уменьшить, стабилизировать). Если предложенный список не содержит нужного действия, пользователь может ввести новое.

Пользователь выбирает параметр одного из объектов, изменение которого приводит к нужному изменению параметра системы.

Если такого параметра нет ни у одного из объектов, пользователь ставит соответствующую отметку.

Шаг 7. В зависимости от действий пользователя на экран выводится нужное описание и иллюстрация стандарта или стандартов и вопрос о том, решена ли задача. Если пользователь считает, что задача не решена, предлагается перейти к Шагу 2 или использованию АРИЗ-82 или его модификаций.

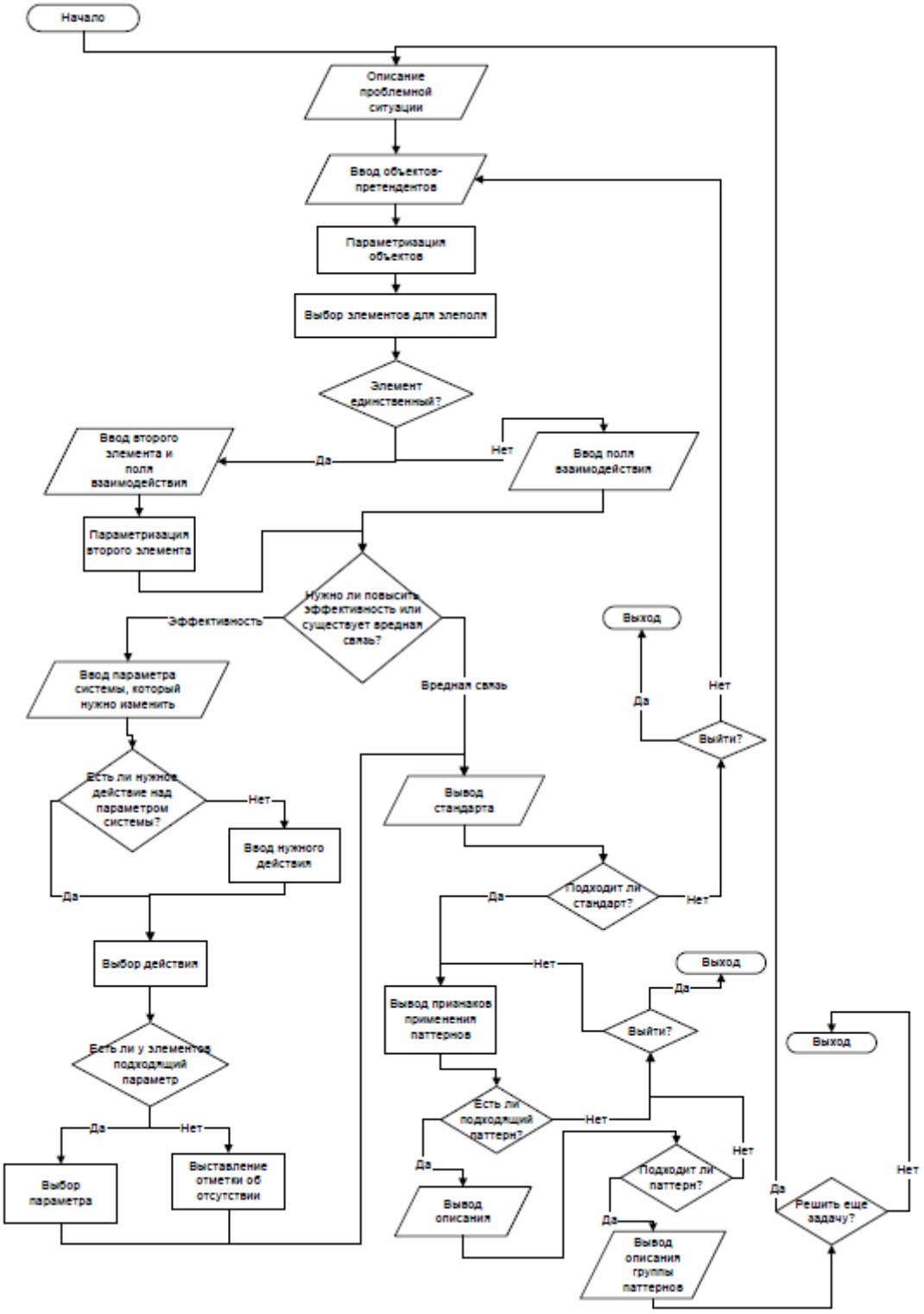
Шаг 8. В зависимости от результатов подбора стандарта или группы стандартов пользователю предлагается список описаний ситуаций применения паттернов. Если не удается найти подходящую ситуацию, предлагается перейти к полному списку паттернов. Если не удается подобрать подходящий паттерн предполагается, что задача не должна решаться с помощью паттернов и предлагается выход из приложения.

Шаг 9. Если удалось подобрать подходящий паттерн, выводится подробное его описание и иллюстрация. Если по описанию паттерн не подходит, предлагается перейти к Шагу 8, родственным паттернам или выйти из приложения.

Шаг 10. Если по описанию паттерн подходит, предлагается перейти к указаниям использования этого паттерна совместно с другими паттернами или выйти из приложения.

Шаг 11. Выводится описание группы паттернов, предлагается решить еще одну задачу или выйти из приложения.

Блок-схема алгоритма:



Приложение содержит систему справки, предоставляющую необходимую теоретическую информацию на каждом шаге алгоритма. Предоставляется возможность сохранения текущего состояния.

Программное приложение предназначено для выполнения следующих функций:

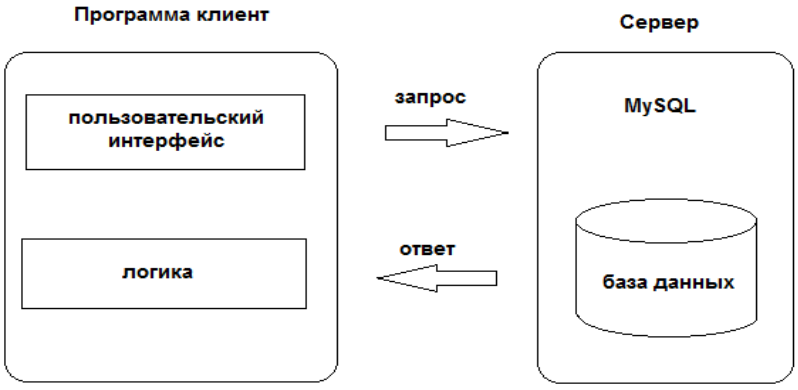
1) обучение на примерах решению задач с помощью стандартов ТРИЗ;
2) изучение стандартов ТРИЗ;
3) изучение паттернов проектирования;
4) поиск решений для проблемных ситуаций из области программирования с помощью стандартов ТРИЗ и шаблонов.

Поэтому, с одной стороны, приложение представляет собой экспертную систему с неизменяемой (на данный момент) базой знаний для интерактивного общения с пользователем для поиска решений. С другой - обучающую программу, на основе которой можно изучить механизм применения стандартов ТРИЗ и сами стандарты, а также паттерны проектирования.

В связи с этим существует два способа работы пользователя с программой: обучающий режим позволяет на примере проследить процесс решения задачи, режим решения дает возможность редактировать поля для ввода информации, касающейся задачи.

3.2 Описание архитектуры разрабатываемой системы

Архитектура системы представляет собой двухуровневое клиент-серверное приложение, где и клиентская часть, и серверная расположены на одном компьютере:



База данных содержит описательную информацию по стандартам и паттернам, хранящуюся в ячейках соответствующих таблиц.

Приложение, являясь клиентом, отправляет сформулированные на языке SQL запросы к СУБД на получение данных из полей таблиц.

Обработка данных происходит на сервере, результат отправляется клиенту.

Компоненты, отвечающие за пользовательский интерфейс и прикладную логику, размещаются на стороне клиента и образуют один уровень.

Прикладная логика отвечает за принципы и зависимости поведения объектов при работе системы, представляет собой набор правил, согласно которым совокупность действий пользователя приводит к выполнению требуемой задачи. На уровне прикладной логики реализованы правила, описанные с помощью основного и вспомогательных алгоритмов, взаимодействия элементов пользовательского интерфейса, правила и зависимости поведения объектов модели предметной области.

Логика отвечает за доступ и работу с данными из базы данных: соединение с сервером, отправку запросов, обработку результатов.

3.3 Описание интерфейса системы

Интерфейс приложения отражает совокупность выполняемых этапов алгоритма, реализованных с учетом удобства пользователя. Для этого, с одной стороны, каждый шаг подразумевает интерактивное взаимодействие с человеком и отражает суть выполнения в естественном виде. С другой - интерактивная функция графического интерфейса, которая представляет собой последовательно сменяющие друг друга диалоговые окна для выполнения задачи, разбитой на этапы, позволяет воспринимать приложение как модель поведения эксперта в определенной области знаний с использованием процедур логического вывода и принятия решений.

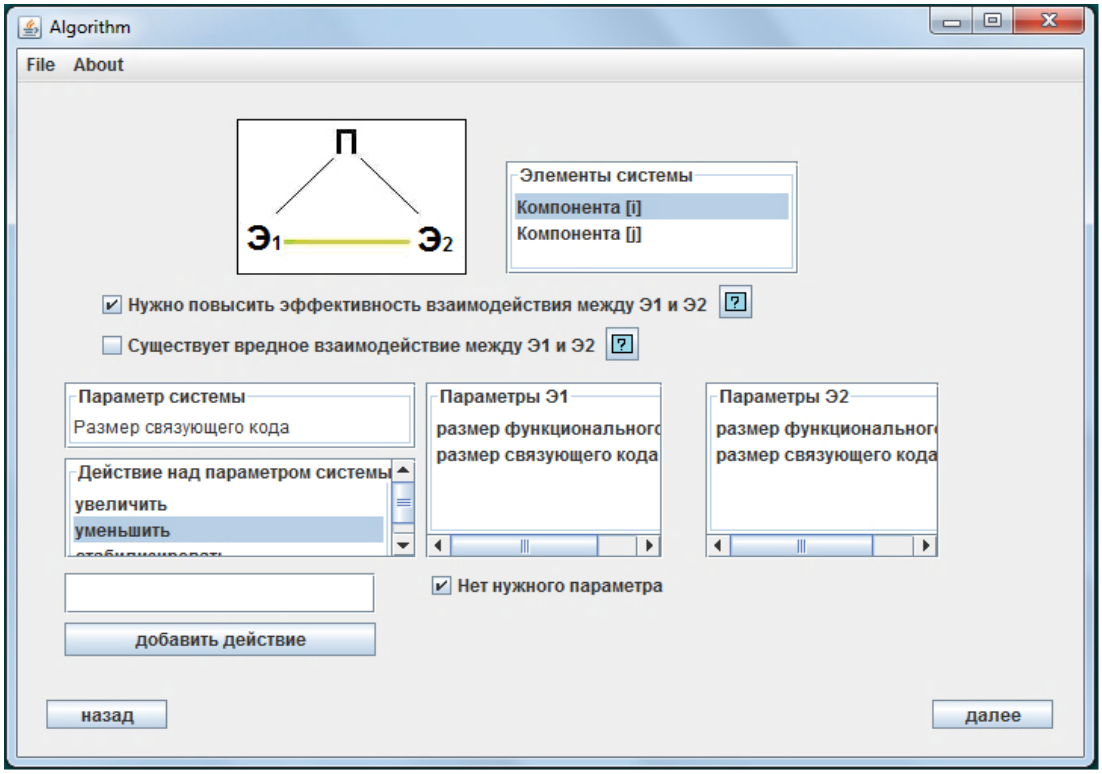
Одним из основных требований к интерфейсу стала предсказуемая работа системы, чтобы пользователь заранее интуитивно понимал, какое действие выполнит программа после ввода данных и выполнения действий.

Интерфейс системы состоит из графических компонентов библиотеки Swing. В основе лежит контейнер формы, в который помещены элементы-панели. Каждый шаг алгоритма, как правило, соответствует некоторой панели, объединяющий компоненты пользовательского интерфейса: кнопки, текстовые поля, текстовые области, списки, другие панели, метки и пр.

Основными компонентами, с которыми имеет дело пользователь, являются кнопки, текстовые области, списки и переключатели. С их помощью реализуются основные процессы в работе алгоритма: пользователь вводит новые или манипулирует уже введенными данными, прямо или косвенно определяет ветви работы алгоритма и пр.

Пример диалогового окна:

Шаг 6. Определение типа задачи



3.4 Описание технологий и инструментов

При написании приложения используется язык Java. Являясь распространенным объектно-ориентированным языком, позволяет реализовать архитектуру и обеспечить возможность дальнейшего развития приложения.

Для создания, модификации и управления данными в базах данных используется информационно-логический язык SQL. Из его преимуществ можно выделить основные: независимость от конкретной СУБД и декларативность.

Разработка программного приложения ведется на базе интегрированной среды разработки IDE NetBeans. Это среда с открытым кодом, которая предоставляет возможность создания сложного пользовательского интерфейса и поддерживает современные технологии. В состав пакета среды IDE входит драйвер для сервера базы данных MySQL, а также встроенный редактор SQL, который позволяет управлять данными из среды.

Для организации хранилища и работы с данными используется система управления базами данных MySQL, используется в качестве сервера, к которому обращается в качестве клиента приложение.

Используемые библиотеки для создания графического интерфейса: AWT и Swing. Положительная сторона компонентов Swing - универсальность интерфейса созданных приложений на всех платформах.

Заключение

Эффективность от уместного применения паттернов проектирования и стандартов ТРИЗ для области проектирования и технической области соответственно не вызывает сомнений.

В процессе исследования была достигнута основная цель, заключающаяся в возможности решать задачи программирования с помощью обоих этих инструментов.

Для чего мною был решен ряд следующих задач.

Во-первых, новая система стандартов ТРИЗ была адаптирована для области программирования.

Во-вторых, для того, чтобы расширить спектр задач, решаемых с помощью паттернов, была сделана попытка переформулировки шаблонов в терминах новых элементов - объединений классов и объектов, рассматриваемых как одно целое.

В-третьих, был разработан единый алгоритм АИСТ-2010-П решения задач из области программирования на основе АИСТ-2010 с расширением на выбор паттернов.

На основе этого алгоритма создано программное приложение, представляющее собой справочно-обучающую систему, позволяющую решать задачи из области программирования с помощью стандартов ТРИЗ и паттернов проектирования.

Также, была собрана картотека задач, на основе которой проверялась работа алгоритма и вносились изменения и уточнения.

Эта система может быть использована как инструмент для изучения стандартов ТРИЗ и алгоритма их применения, так и как небольшая справочная система при освоении паттернов проектирования, т.к. содержит их полный список с описанием структуры, механизма использования и признаков применения.

Результаты данной дипломной работы легли в основу одного из разделов (о решении задач в программировании при помощи стандартов паттернов) статьи [8].

Список литературы

1. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб.: Питер, 2001.

2. Крэг Ларман. Применение UML 2.0 и шаблонов проектирования.- М.: Вильямс, 2008.

3. Одинцов И.О., Рубин М.С. Повышение эффективности разработки программных продуктов на основе методов ТРИЗ / Научно-практическая конференция «ТРИЗ-Фест 2009»: сборник трудов конференции. СПб., 2009.

4. Рубин М.С., О новой системе стандартов на решение изобретательских задач, 2009, http://www.temm.ru/ru/section.php?docId=4201.

5. Рубин М.С., Алгоритм применения стандартов на решение изобретательских задач - 2010, http://www.temm.ru/ru/section.php?docId=4423.

6. Рубин М.С., Параметрический ТРИЗ, 2009, http://www.temm.ru/ru/section.php?docId=4466.

7. Майкл Морган. Java 2 Руководство разработчика. – М.: Вильямс, 2000.

8. Рубин М.С., Одинцов И.О., Пономарева А.В., Зиненко О.И., Прогнозирование развития программного обеспечения на основе ТРИЗ, 2010, http://www.triz-summit.ru/ru/section.php?docId=4625.