Федеральное агентство по образованию Российской Федерации
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет информатики
Кафедра прикладной информатики

УДК 681.03

ДОПУСТИТЬ К ЗАЩИТЕ В ГАК
Зав. кафедрой, проф., д.т.н.
_____ С.П. Сущенко
«___» _____ 2005 г.

Малиновский Антон Юрьевич

МЕТОД ОПИСАНИЯ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ С ПОМОЩЬЮ МОДИФИЦИРОВАННЫХ SADT ДИАГРАММ

Дипломная работа

Научный руководитель, О.А. Змеев
доцент, к.т.н.

Исполнитель, А.Ю. Малиновский
студ. гр. 1401

Электронная версия дипломной работы помещена
в электронную библиотеку. Файл
Администратор

Томск - 2005

Реферат

Дипломная работа 50 с., 29 рис., 12 источников, 2 прил.

ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ, SADT, IDEF0, USE CASE, МОДЕЛЬ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ, МОДЕЛЬ ПРЕЦЕДЕНТОВ, ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ, АРХИТЕКТУРА, JAVA, SWING

Объект исследования: метод описания вариантов использования с помощью модифицированных SADT диаграмм.

Цель работы: разработка метода описания вариантов использования с помощью модифицированных SADT диаграмм, построение прототипа расширяемого инструмента SADT моделирования.

Метод исследования: эксперимент

Результат работы: разработан метод описания вариантов использования с помощью модифицированных SADT диаграмм, предложена нотация являющаяся модификацией SADT нотации, разработан прототип расширяемого инструмента SADT моделирования.

Основные характеристики: Разработанный метод позволяет объединить преимущества методологии структурного проектирования SADT и вариантов использования. Разработанная архитектура ядра инструмента SADT проектирования позволяет создавать модификации графического языка структурного проектирования, который он использует

Область применения: метод готов к использованию и применяется для описания функциональных требований к системам, готов прототип расширяемого инструмента SADT проектирования.

Содержание

Введение

Начальным этапом проектирования любой, в том числе и программной, системы является выделение функциональных требований. Они служат базисом при проектировании системы и дают общее представление о ее функциональности заказчику. От простоты чтения модели функциональных требований часто зависит судьба будущего проекта, поскольку вопрос о его финансировании решается сразу после подготовки модели [1].

Необходимость определения требований к информационной системе возникает в нескольких случаях: в момент выбора новой информационной системы, при подготовке тендерной документации, при заключении договора на разработку или настройку выбранной информационной системы, при уточнении (детализации) потребностей бизнеса в процессе разработки или настройки системы, а так же при необходимости внесения изменений в систему в ходе эксплуатации. На основе выделенных требований в последствие делается заключение о сроках и стоимости разработки или модификации системы. В последствие функциональные требования к системе становятся критерием оценки степени завершенности и корректности исполнения системы. Функциональные требования являются договором между заказчиком информационной системы и ее разработчиком независимо от того, относятся ли разработчик и заказчик системы к одной организации или к разным организациям, состоящим в отношениях заказчик – подрядчик. Такое положение функциональных требований делает их ядром любой разрабатываемой системы. И зачастую успех или неуспех продукта зависит от выбора и реализации метода выделения и описания функциональных требований к системе [2].

В каждом случае, когда возникает проблема выбора метода выделения и описания функциональных требований к системе, перед специалистами предприятия и организации встает задача выбора уровня детализации требований, методов описания, включая формализованное описание с использованием графического моделирования. В настоящее время существуют два основных подхода к моделированию функциональных требований. Поэтому чаще всего выбор методов описания функциональных требований к системе сводится к выбору между двумя этими базовыми подходами.

Первый подход основывается на методологии структурного анализа и проектирования SADT (Structured Analysis and Design Technique), эта методология была реализована в виде стандарта IDEF0 (Integration DEFinition). В настоящий момент методология структурного проектирования SADT (в виде стандарта IDEF0, который лежит в основе семейства методологий IDEF) является государственным стандартом США и активно используется для описания функциональных требований в проектах, которые разрабатываются в Министерстве Обороны США. Кроме того, методология SADT широко используется для описания бизнес процессов организаций. Эта методология является классическим подходом к описанию функциональных требований к системам и преподается в большинстве западных университетов.

Второй подход реализован в ряде методологий проектирования, основанных на унифицированном языке моделирования UML (Rational Unified Process – RUP, Microsoft Solutions Framework – MSF и другие) [2]. В этих методологиях для описания

функциональных требований к системе, которые являются их ядром, используются описания, оформленные в виде вариантов использования (Use Cases). В настоящее время большинство компаний, занимающихся разработкой коммерческого программного обеспечения, использует одну из этих методологий для постановки процесса разработки. Соответственно для описания функциональных требований к разрабатываемым системам эти компании используют модели вариантов использования (в русском переводе также встречается термин «модель прецедентов»).

В данной работе предлагается метод, который может объединить эти два основные подхода к моделированию функциональных требований к системе, сохранив при этом лучшие качества каждого из них. Для этого показывается соответствие между диаграммами, созданными с помощью графического языка структурного проектирования SADT, и сценариями вариантов использования. Затем предлагается нотация, являющаяся модификацией SADT-нотации, которая позволяет описывать варианты использования. Также описываются трудности, которые возникают при использовании модифицированной нотации в существующих инструментах SADT-моделирования.

На основании практического опыта внедрения модифицированной SADT нотации обосновывается необходимость создания инструмента SADT-моделирования, который бы позволял создавать расширения графического языка структурного проектирования, использующегося в нем. Приводится список функциональных требований для этого инструмента и описывается архитектура расширяемого ядра инструмента SADT-моделирования, основанная на прототипе инструмента, реализованном посредством языка программирования Java.

1. Краткий обзор методологий, рассматриваемых в работе.

1.1 Методология структурного анализа и проектирования SADT

Методология структурного проектирования SADT возникла в конце 60-х годов в ходе революции, вызванной структурным программированием. В то время, когда большинство специалистов было занято созданием программного обеспечения, немногие старались разрешить более сложную задачу создания крупномасштабных систем, включающих как людей и машины, так и программное обеспечение, аналогичных системам, применяемым в телефонной связи, промышленности и управлении. В то время специалисты, традиционно занимавшиеся созданием крупномасштабных систем, стали осознавать необходимость большей упорядоченности. Таким образом, разработчики начали формализовать процесс создания системы, разбивая его на следующие фазы[1]:

- анализ - определение того, что система будет делать,
- проектирование - определение подсистем и их взаимодействие,
- реализация - разработка подсистем по отдельности, объединение - соединение подсистем в единое целое,
- тестирование - проверка работы системы,
- установка - введение системы в действие,
- функционирование - использование системы.

Эта последовательность всегда выполнялась итерационно, потому что система полностью никогда не удовлетворяла требованиям пользователей, поскольку их требования часто менялись. И, тем не менее, с этой моделью создания системы, ориентированной на управление, постоянно возникали сложности. Эксплуатационные расходы, возникавшие после сдачи системы, стали существенно превышать расходы на ее создание и продолжали расти с огромной скоростью из-за низкого качества исходно созданной системы. Некоторые считали, что рост эксплуатационных расходов обусловлен характером ошибок, допущенных в процессе создания системы. Исследования показали, что большой процент ошибок в системе возник в процессе анализа и проектирования, гораздо меньше их было допущено при реализации и тестировании, а цена (временная и денежная) обнаружения и исправления ошибок становилась выше на более поздних стадиях проекта. Например, исправление ошибки на стадии проектирования стоит в 2 раза, на стадии тестирования - в 10 раз, а на стадии эксплуатации системы - в 100 раз дороже, чем на стадии анализа. На обнаружение ошибок, допущенных на этапе анализа и проектирования, расходуется примерно в 2 раза больше времени, а на их исправление - примерно в 5 раз, чем на ошибки, допущенные на более поздних стадиях [1]. Кроме того, ошибки анализа и проектирования обнаруживались часто самими пользователями, что вызывало их недовольство.

Традиционные подходы к созданию систем приводили к возникновению многих проблем. Не было единого подхода. Привлечение пользователя к процессу разработки не контролировалось. Проверка на согласованность проводилась нерегулярно или вообще отсутствовала. Результаты одного этапа не согласовывались с результатами других. Процесс с трудом поддавался оценкам, как качественным, так и количественным [1]. Утверждалось, что когда создатели систем пользуются методологиями типа структурного программирования и проектирования сверху вниз, они решают либо не поставленные задачи, либо плохо поставленные, либо хорошо поставленные, но неправильно понятые

задачи. Кроме того, ошибки в создании систем становились все менее доступны выявлению с помощью аппаратных средств или программного обеспечения, а наиболее катастрофические ошибки допускались на ранних этапах создания системы. Часто эти ошибки были следствием неполноты функциональных спецификаций или несогласованности между спецификациями и результатами проектирования. Проектировщики знали, что сложность систем возрастает и что определены они часто весьма слабо. Рост объема и сложности систем является жизненной реалией. Эту предпосылку нужно было принять как неизбежную. Но ошибочное определение системы не является неизбежным: оно - результат неадекватности методов создания систем. Вскоре был выдвинут тезис: совершенствование методов анализа есть ключ к созданию систем, эффективных по стоимости, производительности и надежности. Для решения ключевых проблем традиционного создания систем широкого профиля требовались новые методы, специально предназначенные для использования на ранних стадиях процесса. Применение SADT проистекало из этого убеждения. Методы, подобные SADT, на начальных этапах создания системы позволяли гораздо лучше понять рассматриваемую проблему. А это сокращает затраты как на создание, так и на эксплуатацию системы, а кроме того, повышает ее надежность. SADT - это способ уменьшить количество дорогостоящих ошибок за счет структуризации на ранних этапах создания системы, улучшения контактов между пользователями и разработчиками и сглаживания перехода от анализа к проектированию [1].

SADT (аббревиатура выражения Structured Analysis and Design Technique - методология структурного анализа и проектирования) - это методология, разработанная специально для того, чтобы облегчить описание и понимание искусственных систем, попадающих в разряд средней сложности. SADT была создана и опробована на практике в период с 1969 по 1973 г. Эта методология возникла под сильным влиянием PLEX, концепции клеточной модели человек-ориентированных функций Хори, общей теории систем, технологии программирования и даже кибернетики. С 1973 г. сфера ее использования существенно расширяется для решения задач, связанных с большими системами, такими, как проектирование телефонных коммуникаций реального времени, автоматизация производства (CAM), создание программного обеспечения для командных и управляющих систем, поддержка боеготовности. Она с успехом применялась для описания большого количества сложных искусственных систем из широкого спектра областей (банковское дело, очистка нефти, планирование промышленного производства, системы наведения ракет, организация материально-технического снабжения, методология планирования, технология программирования). Причина такого успеха заключается в том, что SADT является полной методологией для создания описания систем, основанной на концепциях системного моделирования [1].

Под моделью в SADT понимают описание системы (текстовое и графическое), которое должно дать ответ на некоторые заранее определенные вопросы. Процесс моделирования какой-либо системы в SADT начинается с определения контекста, т.е. наиболее абстрактного уровня описания системы в целом. В контекст входит определение субъекта моделирования, цели и точки зрения на модель. Под субъектом понимается сама система, при этом необходимо точно установить, что входит в систему, а что лежит за ее пределами, другими словами, мы должны определить, что мы будем в дальнейшем рассматривать как компоненты системы, а что как внешнее воздействие. На определение субъекта системы будет существенно влиять позиция, с которой рассматривается система, и цель моделирования - вопросы, на которые построенная модель должна дать ответ. Модель в SADT предполагает наличие четко сформулированной цели, единственного субъекта моделирования и одной точки зрения.

Основу методологии SADT составляет графический язык описания бизнес-процессов. Модель в нотации SADT представляет собой совокупность иерархически

упорядоченных и взаимосвязанных диаграмм. Вершина этой древовидной структуры, представляющая собой самое общее описание системы и ее взаимодействия с внешней средой, называется контекстной диаграммой. После описания системы в целом проводится разбиение ее на крупные фрагменты. Этот процесс называется функциональной декомпозицией, а диаграммы, которые описывают каждый фрагмент и взаимодействие фрагментов, называются диаграммами декомпозиции. После декомпозиции контекстной диаграммы проводится декомпозиция каждого большого фрагмента системы на более мелкие и так далее до достижения нужного уровня подробности описания. После каждого сеанса декомпозиции проводятся сеансы экспертизы - эксперты предметной области указывают на соответствие реальных бизнес - процессов созданным диаграммам. Найденные несоответствия исправляются и только после прохождения экспертизы без замечаний можно приступать к следующему сеансу декомпозиции. Таким образом, достигается соответствие модели реальным бизнес - процессам на любом и каждом уровне модели. Синтаксис описания системы в целом и каждого ее фрагмента одинаков во всей модели.

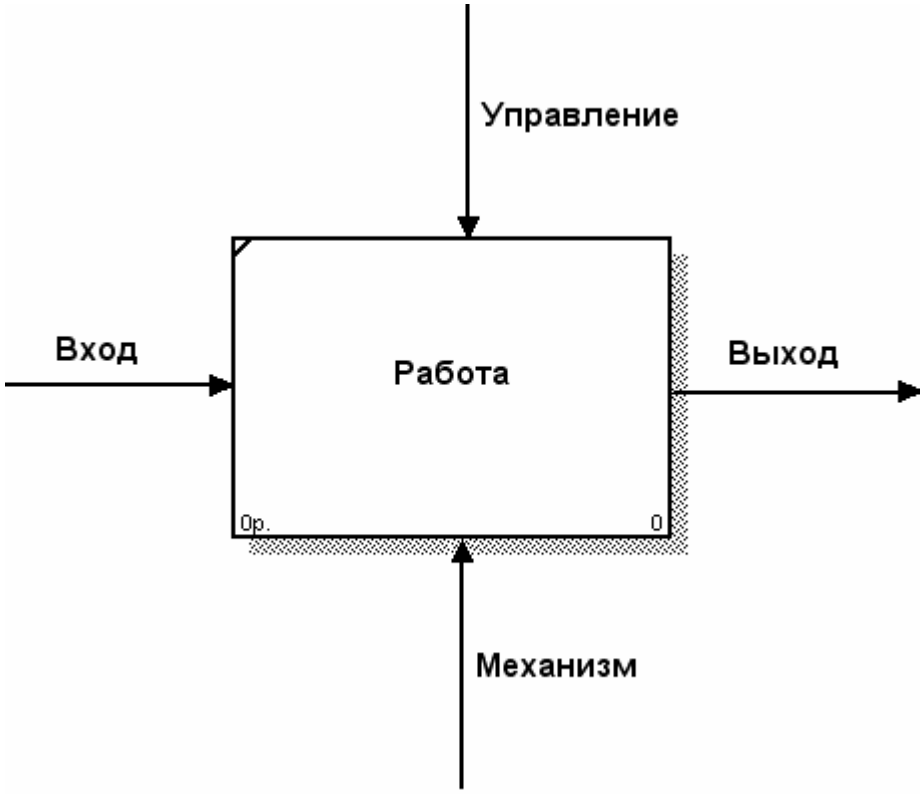

Рис. 1. Единица графического языка SADT

Работы (Activity), которые означают некие поименованные процессы, функции или задачи, изображаются в виде прямоугольников. Именем работы должен быть глагол или глагольная форма (например "Изготовление детали", "Прием заказа" и т.д.). Взаимодействие работ с внешним миром и между собой описывается в виде стрелок. Стрелки представляют собой некую информацию и именуются существительными (например, "Заготовка", "Изделие", "Заказ"). В SADT различают четыре типа стрелок [1]:

Вход (Input) - материал или информация, которая используется или преобразовывается работой.

Управление (Control) - правила, стратегии, процедуры или стандарты, которыми руководствуется работа. Каждая работа должна иметь хотя бы одну стрелку управления.

Выход (Output) - материал или информация, которая производится работой. Каждая работа должна иметь хотя бы одну стрелку выхода.

Механизм (Mechanism)- ресурсы, которые выполняют работу, например, персонал предприятия станки, механизмы и т.д.

Каждый тип стрелок подходит или выходит к определенной стороне прямоугольника, изображающего работу. К левой стороне подходят стрелки входов, к верхней - стрелки управления, к нижней - механизмов реализации выполняемой функции, а из правой - выходят стрелки выходов. Такое соглашение предполагает, что, используя управляющую информацию и реализующий ее механизм, функция преобразует свои входы в соответствующие выходы.

1.2 Варианты использования

Понятие «Вариант использования» появилось в середине восьмидесятых годов благодаря шведскому ученому Айвару Якобсону, который ввел шведский термин «anvendningfall», что в приблизительном переводе означает «ситуация использования» или, по-английски, «usage case». В последствие этот термин был изменен на «use case», поскольку такое звучание было более близко английскому языку [3].

Варианты использования явились результатом 20-ти летней работы, которую Якобсон вел для компании Eriksson, руководя разработкой сложного программного обеспечения [4]. Варианты использования были разработаны в качестве почти универсального средства для определения функциональных требований к любым программным системам, в том числе и собираемым из компонентов, но с их помощью можно не только определять требования. Варианты использования направляют весь процесс разработки программного обеспечения, они вносят значительный вклад в поиск и описание классов, подсистем и интерфейсов, а также поиск и описание вариантов тестирования, планирование итераций разработки и системной интеграции [5].

Создание приложения в ходе каждой итерации, кроме самой первой, на которой и происходит первоначальное выделение вариантов использования, направляется вариантами использования при проходе по всем рабочим процессам, от определения функциональных требований к системе до ее проектирования и тестирования. Каждое приращение разработки, таким образом, возникает в результате работы по реализации набора вариантов использования. Другими словами, в каждой итерации разработки программного обеспечения определяется и реализовывается некоторое количество вариантов использования. Варианты использования, наиболее сильно влияющие на архитектуру разрабатываемой системы, называются архитектурно значимыми вариантами использования и реализуются в первую очередь. Тем самым достигается стабильная архитектура, которая используется на протяжении множества последующих циклов разработки [5].

Каждый вариант использования определяет один из вариантов взаимодействия пользователей с системой, поэтому они являются идеальной отправной точкой для разработки руководства пользователя и создания приемочных тестов, которые оценивают реализацию системы с точки зрения пользователя.

Оценивая, как часто выполняются различные варианты использования, можно определить, какой из них требует особой эффективности. Эта оценка может быть использована для определения требуемой производительности процессора используемого компьютера или оптимизации структуры базы данных под определенные операции [5]. Кроме того, оценка частоты выполнения вариантов использования позволяет создать серию нагрузочных тестов, которые оценивают поведения системы в целом при пиковых нагрузках. Особенно часто это свойство вариантов использования применяется при создании распределенных приложений. Также подобные оценки могут быть применены для повышения удобства работы с системой, становится возможным выделить наиболее важные варианты использования для того, чтобы разрабатывать их пользовательские интерфейсы особенно тщательно.

Обычно в сложной системе имеется несколько категорий пользователей. Каждая категория пользователей представляется в варианте использования в виде действующего лица, которое использует систему, участвуя в вариантах использования [4]. Вариант использования – это последовательность действий, которая система предпринимает для

получения результата ценного для действующего лица. Полный набор действующих лиц системы и вариантов использования образует модель вариантов использования. Более формальное определение варианта использования дают А. Якобсон, Г. Буч, Дж. Рамбо в одной из основополагающих книг объектно-ориентированного проектирования «Унифицированный процесс разработки программного обеспечения»[5]:

«Вариант использования определяет последовательность действий (включая ее варианты), которые может выполнить система, приносящих ощутимый и измеримый результат, значимый для некоторого действующего лица».

Варианты использования могут быть записаны с помощью различных средств (графических – таких как диаграммы последовательностей, диаграммы взаимодействий и текстовых) Разработчики вариантов использования чаще всего предпочитают описывать варианты использования в текстовом виде, справедливо полагая, что диаграммы взаимодействий (Collaboration diagram) и диаграммы состояний (Statechart diagram) ненаглядны и не имеют всей функциональности текстового описания варианта использования, кроме того, в этих диаграммах затруднительно описывать расширения вариантов использования. Диаграммы вариантов использования (Use Case diagram), которые имеются в языке UML, применимы только для описания отношения между вариантами использования и отображения связи действующих лиц и вариантов использования [4].

Существует несколько общепринятых форматов записи вариантов использования в текстовом виде, однако практически в любом формате вариант использования состоит из нескольких основных частей [4]:

- действующее лицо - участник, который обращается к системе за одной из ее услуг
- цель - то, чего пытается добиться от системы действующее лицо
- предусловия и гарантии - условия, которые должны быть истинными на момент активации варианта использования и после его окончания
- основной сценарий – набор последовательных шагов в варианте использования, при отсутствии ошибок
- расширения – отклонения от основного сценария варианта использования

Пример описания варианта использования в текстовой форме:

Вариант использования «Создание курса»

Автор предоставляет необходимую информацию для создания страницы курса, Система создает страницу курса, где отображена предоставленная информация.

Область действия: система учета учебной литературы

Основное действующее лицо: Автор

Участники и их интересы:

- Автор желает создать страницу своего курса
- Система создает страницу курса

Предусловия:

- Система корректно установлена и работает
- Сеанс взаимодействия Автора и Системы не прерывается

- Автор аутентифицирован
- Диалог создания курса отображается у Автора

Минимальные гарантии:

- Система создаст страницу с предоставленной информацией

Гарантия успеха: Система создаст страницу с предоставленной информацией

Триггер: действие Автора

Основной сценарий:

1. Автор вводит информацию о курсе
2. Автор выбирает ссылки на книги из хранилища
3. Автор инициирует создание страницы
4. Система проверяет время существования сессии с Автором
5. Система создает страницу с предоставленной информацией
6. Система уведомляет Автора о создании страницы

Альтернативы:

4.1 Система обнаруживает, что превышен лимит существования аутентифицированной сессии

4.1.1 Система уведомляет Автора об истечении лимита существования его сессии

4.1.2 Система уничтожает сессию с Автором

4.1.3 Система заносит в журнал сведения об уничтожении сессии

4.1.4 Система создает новую сессию с Автором и присваивает ему права Гостя

4.1.5 Система заносит в журнал сведения о созданной сессии

2. Методология, объединяющая SADT-методологию и варианты использования.

2.1 Сильные и слабые стороны методологий SADT и UC

Методологии структурного анализа и проектирования SADT и варианты использования являются самыми распространенными подходами к описанию функциональных требований к системам, поэтому сравнению этих методологий посвящено множество исследований. В основном исследователи приходили к выводу, что семейство стандартов IDEF лучше подходит для анализа и моделирования бизнес процессов, в то время как методологии, основанные на вариантах использования, скорее ориентированы непосредственно на разработку программного обеспечения [6].

Свое широкое распространение методология SADT получила, поскольку модели функциональных требований, построенные с помощью ее графического языка, сочетают в себе такие трудно совместимые качества, как легкость чтения (неподготовленный пользователь начинает свободно понимать модели, построенные в SADT-нотации примерно после 30-ти минут подготовки) и сильный формализм, который позволяет проводить стоимостный и временной анализы, функциональных требований к системе. Иерархическое представление модели позволяет легко увидеть полную картину системы, что сильно упрощает добавление и модификацию функциональных требований к системе. Ограниченность в изобразительных средствах и достаточно четкая определенность графического языка проектирования SADT, способствуют однозначному пониманию модели разными людьми.

Недостатком этой методологии проектирования является ее изолированность от современных методологий разработки программного обеспечения, поскольку семейство методологий IDEF, хотя и предоставляет мощные средства для моделирования функциональных требований и информационную структуру системы, однако не имеет достаточно развитых средств для управления процессом разработки программного обеспечения на дальнейших этапах. Этим объясняется крайне малое распространение методологий семейства IDEF среди компаний занимающих разработкой программного обеспечения.

Также к недостаткам SADT-проектирования можно отнести слишком общее определение примитивов моделирования, что, в некоторых случаях, ведет к различным их интерпретациям, а, следовательно, затрудняет моделирование и чтение модели. Так, например, активатором работы, в соответствие с методологией является элемент «вход» [8], однако часто встречаются ситуации, когда работу активирует ее исполнитель, который должен быть отнесен к «механизмам» системы. Однако все же основной проблемой является именно изолированность методологии SADT от современных процессов разработки программного обеспечения, поскольку именно этот фактор значительно ограничивает ее применимость.

Напротив, варианты использования в настоящее время являются основой большинства современных методологий разработки программного обеспечения и широко применяются именно для коммерческой разработки информационных систем. Варианты использования тесно интегрированы с архитектурой системы, моделью тестирования

системы, требованиями к пользовательскому интерфейсу и с созданием руководства пользователя [5].

В свою очередь недостатком подхода, основанном на вариантах использования, является то, что описание функциональных требований к системе с помощью вариантов использования не учитывает входные и выходные данные, а также информационные связи между вариантами использования. Соответственно отсутствует единая картина взаимодействующих функций, модель скорее представляет собой структурированный список описаний [7].

Как видно из сравнения достоинства и недостатки подходов, основанных на вариантах использования и на методологии структурного анализа и проектирования SADT, компенсируют друг друга. Именно этот факт послужил отправной точкой для разработки метода описания функциональных требований к системе, который объединяет достоинства этих методологий и по возможности не имеет их недостатков.

2.2 Выделение вариантов использования из SADT-диаграмм

2.2.1 Необходимость выделения вариантов использования из SADT-диаграмм

Как было показано в предыдущей главе, ни один из существующих подходов к разработке программного обеспечения не лишен существенных недостатков. Так разработка модели вариантов использования осложняется отсутствием модели бизнес-процессов, а разработка функциональных требований к программному обеспечению с помощью методологии SADT осложнена отсутствием хорошо проработанной схемы продолжения цикла разработки, поскольку семейство методологий IDEF имеет многочисленные недостатки, при использовании их для поддержки процесса разработки программного обеспечения.

Существует метод моделирования бизнес процессов, использующий процессный подход аналогичный SADT – метод Ericsson – Penker [9]. Авторы создали свой профиль UML для моделирования бизнес процессов. Основной диаграммой, используемой в этом подходе, является диаграмма деятельности (Activity diagram), которую авторы расширили с помощью механизма стереотипов. В результате получился процесс, в котором не применяются варианты использования, а отправной точкой для моделирования системы служит модель функциональных требований, построенная с помощью метода Ericsson – Penker. Однако это решение не получило популярность в среде разработчиков, поскольку механизм вариантов использования чрезвычайно удобен и хорошо себя зарекомендовал за годы существования. Кроме того, не была создана методология, которая бы полноценно связывала функциональные требования, описанные с помощью метода Ericsson – Penker с остальным процессом разработки программного обеспечения.

Альтернативой этому подходу может служить использование SADT-нотации для моделирования бизнес процессов, с последующим выделением вариантов использования из модели системы. Такой подход позволяет использовать преимущества SADT-нотации для моделирования бизнес процессов на высоком уровне с последующим уточнением их с помощью вариантов использования, которые позволяют более детально моделировать бизнес процессы, чем это возможно с помощью SADT-нотации. Такое использование этой нотации вполне соответствует ее предназначению – создание легко читаемых моделей функциональных требований к системе на начальном этапе разработки.

Однако, несмотря на преимущества, которые можно получить при использовании SADT-нотации совместно с вариантами использования для поддержки процесса разработки программного обеспечения, данный подход не был сколько-нибудь полно реализован в CASE средствах. Не существует также никаких полноценных средств, позволяющих автоматизировать выделение вариантов использования из модели бизнес процессов в нотации SADT, хотя работа в этом направлении ведется.

Первым коммерческим продуктом, в котором появилась начальная реализация такой возможности – был продукт компании Wizard – ProcessWorks! 6.1, в котором позиционировалась возможность выделения вариантов использования из SADT- диаграмм (IDEF0). На сегодняшний день этот продукт остается единственным коммерческим продуктом, где реализована подобная функциональность.

Для реализации автоматического преобразования SADT-диаграммы в варианты использования создатели продукта воспользовались предположением, что механизм SADT диаграммы однозначно соответствует действующему лицу варианта использования, а деятельность SADT диаграммы также однозначно соответствует самому варианту использования.

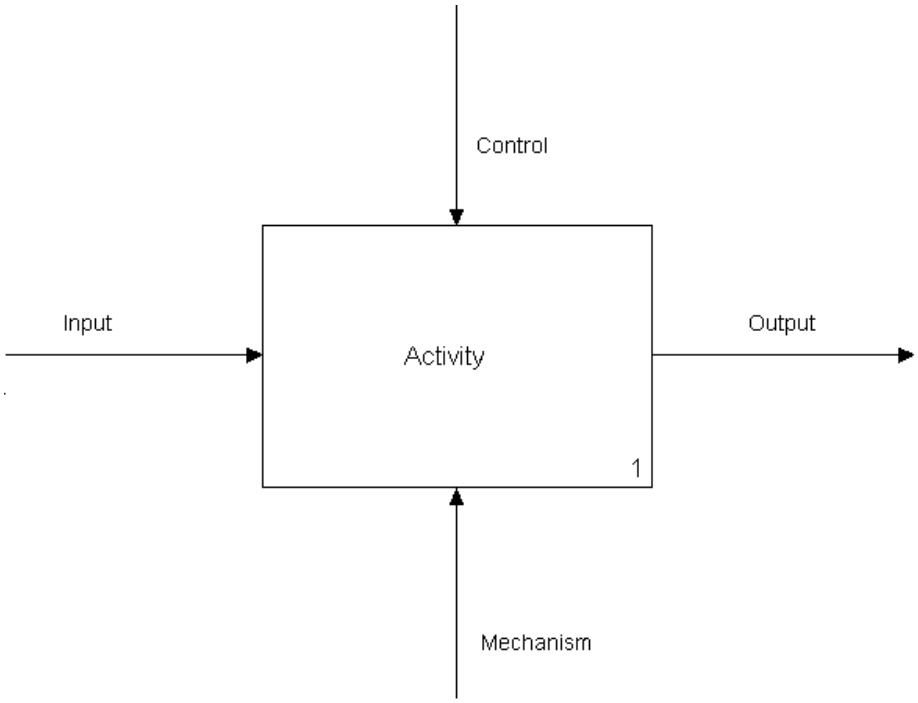

Рис. 2. Простейшее описание деятельности в SADT-нотации

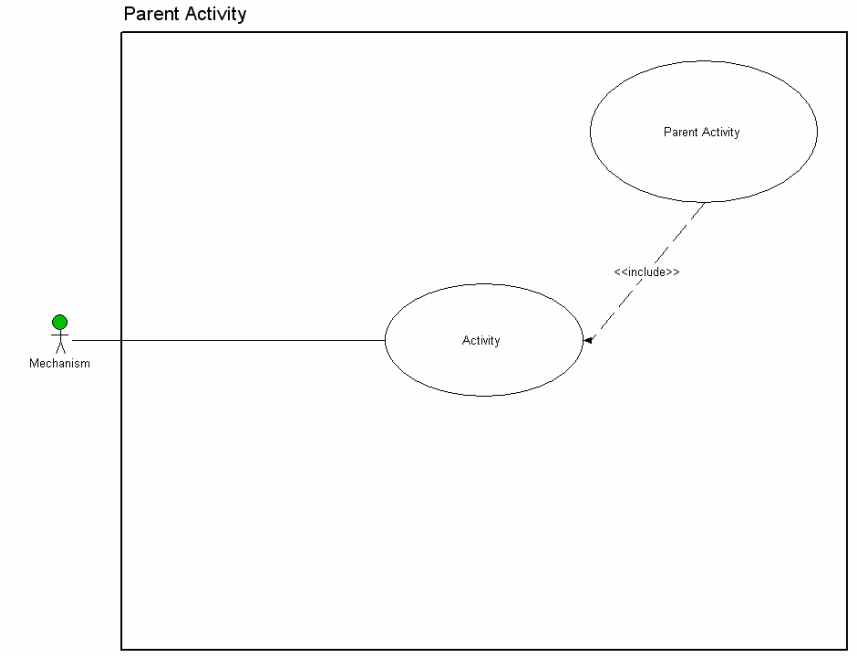

Рис. 3. Модель варианта использования созданная с помощью ProcessWorks! 6.1

Ограничения, которые используются при автоматическом выделении вариантов использования (каждый механизм – действующее лицо, каждая деятельность – вариант использования) и невозможность использования всех элементов диаграммы для создания варианта использования, значительно снижают ценность этой реализации. Однако даже такая реализация автоматического выделения вариантов использования из SADT-диаграмм позиционируется как одно из основных преимуществ этого продукта перед конкурентами, что говорит о востребованности данной возможности среди разработчиков программного обеспечения.

Как было показано ранее, включение построения модели бизнес процессов системы, описанных с помощью SADT-нотации, в процесс разработки программного обеспечения облегчает процесс создания модели вариантов использования. Методология SADT, которая используется для построения моделей бизнес-процессов, хорошо известна системным аналитикам и пользуется среди них большой популярностью. Существует ряд инструментов, которые автоматизируют построение моделей в нотации SADT, при этом существует единый формат хранения SADT-диаграмм, который поддерживается всеми инструментами (IDEF0 IDL). Создание полноценного средства автоматизации выделения вариантов использования из SADT-модели функциональных требований к системе, которое будет иметь возможность работать с уже существующими SADT-моделями, позволит использовать эту методологию в начальной стадии цикла разработки программного обеспечения для создания высокоуровневой модели бизнес процессов. Эта высокоуровневая модель в последствие будет углубляться с помощью модели вариантов использования, представленной в виде набора подробных текстовых описаний вариантов использования. Для того чтобы построение такого автоматизирующего средства стало возможным, необходимо четко и как можно более полно выделить связи между моделью бизнес процессов построенной в нотации SADT и моделью вариантов использования.

2.2.2 Отображение элементов SADT-диаграмм на элементы описания вариантов использования

Основными элементами SADT-диаграмм являются [1]:

- работа – функция или активная часть системы
- вход – ресурсы, которые требуются системе
- выход – ресурсы, которые были созданы в результате преобразования или потребления входных ресурсов
- управление - стандарты и правила, руководящие работой
- механизм – действующие силы и средства, выполняющие работу

Соответствие между SADT-моделями и моделями вариантов использования можно показать, поставив в соответствие этим элементам элементы вариантов использования. Основными элементами вариантов использования являются [11]:

- действующее лицо - участник, который обращается к системе за одной из ее услуг
- цель - то, чего пытается добиться от системы действующее лицо
- предусловия и гарантии - условия, которые должны быть истинными на момент активации варианта использования и после его окончания
- основной сценарий – набор последовательных шагов в варианте использования, при отсутствии ошибок
- расширения – отклонения от основного сценария варианта использования

Отображение элемента «работа»

Работа в нотации SADT отображается на сам вариант использования. При этом отношение вида «декомпозируется» между работами превращается в отношение вида «включает» между соответствующими вариантами использования. Уровень в иерархии диаграмм SADT определяет уровень варианта использования. Работы, которые не

декомпозируются на других диаграммах, являются элементарными вариантами использования или, если говорить в терминологии, применяемой разработчиками вариантов использования, имеют уровень дна. Работа, расположенная на диаграмме самого верхнего уровня, отображается на корневой вариант использования, имеющий самый высокий уровень. Остальные уровни могут быть определены примерно и, в последствие, уточнены автором вариантов использования, осуществляющим преобразование.

Отображение элементов «вход»

Наличие ресурсов для выполнения варианта использования является условием, которое должно быть истинным. Соответственно входы в нотации SADT отображаются на предусловия в варианте использования.

Отображение элементов «выход»

Целью варианта использования является получение одного или нескольких ресурсов, из числа описанных в выходах работы. То есть часть выходов в SADT диаграммах отображается на цели вариантов использования. Оставшиеся выходы представляют собой отдельные элементарные варианты использования, которые должны быть включены в вариант использования, соответствующий работе. В этом случае вариант использования, на который отображается работа, может быть поднят на уровень выше в иерархии и не участвовать в описании потока варианта использования, представленного диаграммой.

Отображение элементов «управление»

Вариант использования, как и работа, должен соответствовать определенным ограничениям и подчиняться определенным правилам, которые описываются в управлении работы. Условие соответствия этим правилам и ограничениям должно быть истинным на протяжении всего времени исполнения варианта использования. Соответственно управление в нотации SADT отображается на элемент «гарантии» в варианте использования.

Отображение элементов «механизм»

Механизмы в нотации SADT делятся на две части: исполнители работы и вспомогательные инструменты. В частности к вспомогательным инструментам относится разрабатываемая система или ее часть. По той части системы, которая служит механизмом для работы, вариант использования, соответствующий этой работе, может быть отнесен к определенному пакету вариантов использования. То есть от механизмов, выполняющих данную работу, зависит то, какой группе разработчиков будет передан соответствующий вариант использования. Механизмы, относящиеся к исполнителям данной работы, должны быть отображены на действующих лиц варианта использования.

Основной сценарий и расширения варианта использования

Поскольку в нотации SADT не определен порядок выполнения работ, то построить пошаговый сценарий возможно только в том случае, если автор, осуществляющий преобразование, явно укажет порядок выполнения работ в системе. После этого можно создать основной сценарий в первом приближении, состоящий из включенных вариантов использования, которые соответствуют работам на декомпозирующей диаграмме.

Поскольку диаграмма в методологии SADT строится без учета возможных сбоев в системе, то, в общем случае, расширения на SADT-диаграмме отсутствуют. Однако автор, выполняющий преобразование, может явно указать работы, а значит и соответствующие им варианты использования, которые будут входить в расширение родительского варианта использования.

2.2.3 Пример преобразования SADT-диаграммы

В качестве примера выделения вариантов использования из SADT диаграммы возьмем диаграмму, которая используется как образец в классической книге по SADT-проектированию «Методология структурного анализа и проектирования SADT». Диаграмма является декомпозицией работы «Изготовить нестандартную деталь».

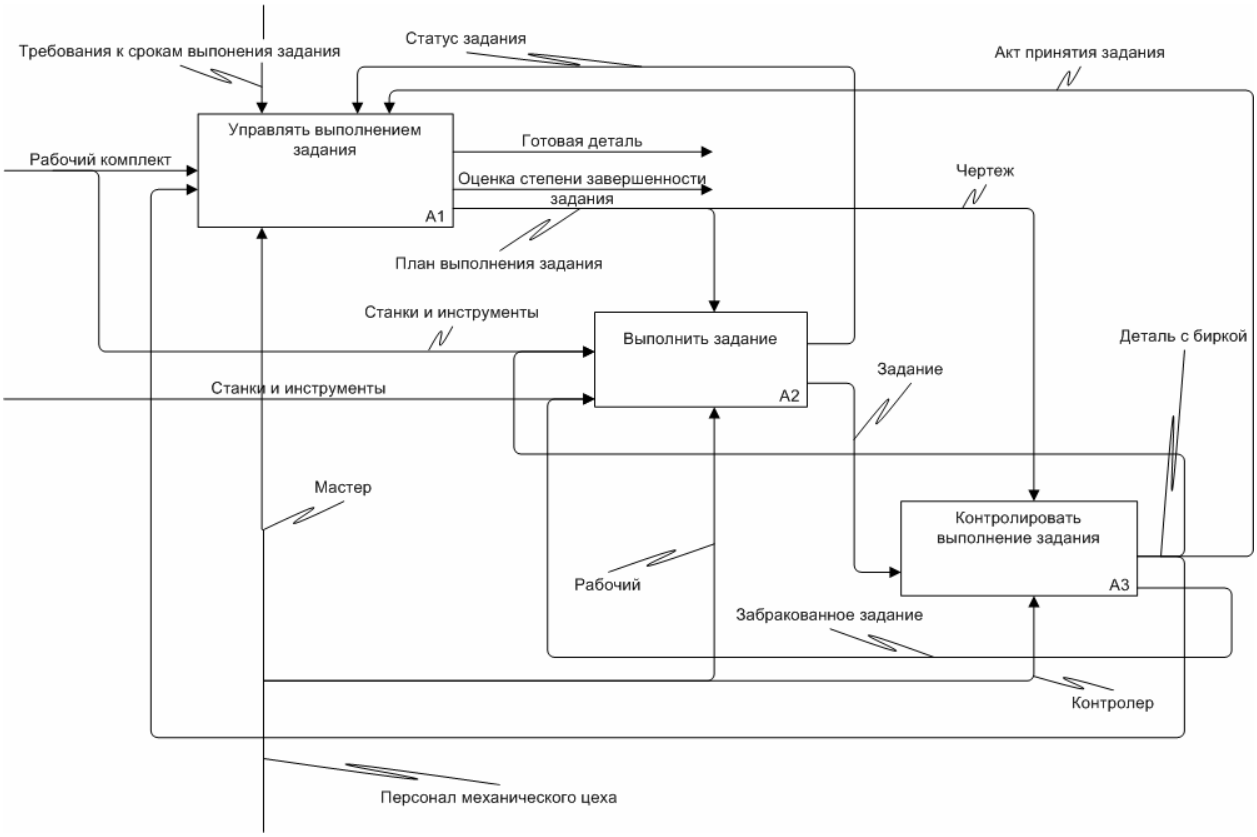

Рис. 4. Диаграмма, декомпозирующая работу «изготовить нестандартную деталь»

Из этой диаграммы с помощью предложенной методики можно выделить три сложных варианта использования, в дальнейшем они могут быть также декомпозированы, и четыре простых варианта использования. Сложные варианты использования: «Управлять выполнением задания», «Выполнить задание», «Контролировать выполнение задания», простые варианты использования: «Принять готовую деталь», «Оценить степень завершенности задания», «Разработать план выполнения задания», «Снабдить деталь биркой», «Отправить забракованное задание на повторное выполнение».

Действующими лицами варианта использования «изготовить нестандартную деталь» являются мастер, рабочий, контролер. Целью этого варианта использования является получение готовой детали. В ходе варианта использования гарантируется то, что срок выполнения задания будет отвечать требованиям к срокам выполнения задания.

Автор, выполняющий преобразование, может также указать варианты использования, относящиеся к основному потоку и к расширениям. В данном случае к основному потоку варианта использования относятся действия «Выполнить задание», «Контролировать выполнение задания», «Разработать план выполнения задания», «Снабдить деталь биркой», «Принять готовую деталь», а к альтернативному потоку варианта использования относится действие «Отправить забракованное задание на

повторное выполнение». Вариант использования «Управлять выполнением задания» декомпозируется на диаграмме своими выходами и выносится на уровень выше в иерархии вариантов использования, в потоках выполнения вариантов использования он не участвует.

Кроме того, для выделения варианта использования необходимо явно задать порядок, в котором выполняются действия на диаграмме, поскольку из SADT-диаграммы эту информацию, в общем случае, извлечь нельзя. Однако спецификация SADT-диаграмм позволяет использовать порядок, в котором находятся деятельности на диаграмме, в качестве порядка выполнения деятельностей.

После окончания процесса выделения варианта использования из диаграммы «Изготовить нестандартную деталь», получившийся вариант использования можно записать с помощью стандартного текстового представления.

Вариант использования «Изготовление нестандартной детали»

Рабочий и Контролер под руководством Мастера изготавливают нестандартную деталь
Основное действующее лицо: Мастер
Участники и их интересы:

- Мастер управляет изготовлением нестандартной детали
- Рабочий выполняет задание Мастера
- Контролер следит за качеством изготавливаемой детали

Предусловия:

- В наличии имеется полный рабочий комплект
- Имеются необходимые станки инструменты для изготовления детали

Минимальные гарантии:

- Срок выполнения задания будет отвечать требованиям к срокам выполнения задания.

Гарантия успеха: Будет изготовлена нестандартная деталь надлежащего качества
Основной сценарий:

1. Мастер разрабатывает план выполнения задания
2. Рабочий выполняет задание Мастера
3. Контролер проверяет качество выполненного задания
4. Контролер снабжает деталь биркой
5. Мастер оценивает степень завершенности задания
6. Мастер принимает готовую деталь

Альтернативы:

3.1 Контролер обнаруживает бракованную деталь
3.1.1 Контролер отправляет задание на повторное выполнение, переход на шаг 2

Рис. 5. Вариант использования, соответствующий диаграмме «Изготовить нестандартную деталь»

Как видно из иллюстрации, функциональное требование к системе, которое задает текстовое представление варианта использования, полностью аналогично функциональному требованию, заданному диаграммой «Изготовить нестандартную деталь».

2.2.4 Требования к редактору SADT-диаграмм, позволяющего осуществлять автоматическое выделение вариантов использования из SADT-диаграмм.

Для того чтобы SADT-модель стала частью процесса разработки программного обеспечения, построенного вокруг вариантов использования, необходимо создать инструмент позволяющий автоматизировать подобное выделение. Исходя из правил преобразования, нетрудно определить требования к инструменту автоматического преобразования.

- Возможность преобразования работы в варианты использования
- Возможность задания уровня работы
- Возможность преобразования входов в предусловия варианта использования
- Возможность преобразования выходов в цели или во включенные варианты использования в зависимости от выбора.
- Возможность преобразования управлений в гарантии вариантов использования
- Возможность преобразования выбранных механизмов в действующих лиц вариантов использования
- Возможность классификации вариантов по используемым механизмам
- Возможность явного задания порядка выполнения работ и создания основного сценарии
- Возможность явного задания работ входящих в расширение варианта использования

Для реализации этих требований необходимо разработать инструмент SADT-моделирования, который позволит изменять внешний вид элементов диаграммы. Также необходимо реализовать возможность получения доступа к примитивам модели, которые содержатся на этой диаграмме, для интеграции это инструмента в процесс разработки программного обеспечения.

2.3 Запись вариантов использования с помощью модифицированных SADT-диаграмм

Как было показано в предыдущей главе, SADT-диаграмма содержит в себе информацию о варианте использования. Однако для преобразования диаграммы в вариант использования требуется участие человека, который должен самостоятельно указывать специфику каждого элемента диаграммы (будет ли эта работа отображаться на вариант использования основного потока или альтернативного; является ли выход целью или декомпозирующим вариантом использования). Соответственно полная автоматизация выделения варианта использования из SADT-диаграммы не представляется возможной [7].

Логическим развитием такого подхода к созданию вариантов использования является изначальная запись диаграммы с помощью специализированных элементов SADT. Этот подход не только упростит переход от представления варианта использования в виде SADT-диаграммы, но и позволит не использовать текстовое представление варианта использования в том случае, если аналитику не требуется слишком подробное его описание. Соответственно необходимо разработать специализированную нотацию SADT диаграмм с тем, чтобы убрать или свести к минимуму элементы неопределенности, которые требуют человеческого участия, а также позволить пользователям легко читать вариант использования по диаграмме.

Для записи вариантов использования с помощью SADT-нотации предлагается использовать нотацию, состоящую из элементов, являющихся расширениями существующих элементов SADT-диаграмм. Поскольку данная методология в настоящее время используется для описания функциональных требований к системам в реальных проектах, то для каждого элемента будет приведено два вида визуализации. На каждой иллюстрации слева (или сверху) будет помещаться вид элемента, который он должен иметь в инструменте SADT-моделирования, поддерживающим расширения нотации, а справа (или снизу) вид, который используется для моделирования с помощью существующих инструментов. Для построения модифицированных SADT-диаграмм в качестве инструмента моделирования использовался продукт компании Computer Associations – AllFusion Process Modeler (ранее известный как BPWin), соответственно все изображения, отражающие существующий инструмент моделирования, были сделаны с помощью этого продукта.

Рис. 6. Деятельность основного потока

Деятельность основного потока представляет собой вариант использования, представляющий один из шагов нормального потока варианта использования, описываемого диаграммой. Номер деятельности основного потока соответствует номеру

шага нормального потока варианта использования, на который он отображается. В классических SADT-диаграммах достигнуть такого соответствия бывает затруднительно, поскольку деятельности там нумеруются согласно их размещению на диаграмме (сверху вниз, слева направо) и месту в иерархии модели.

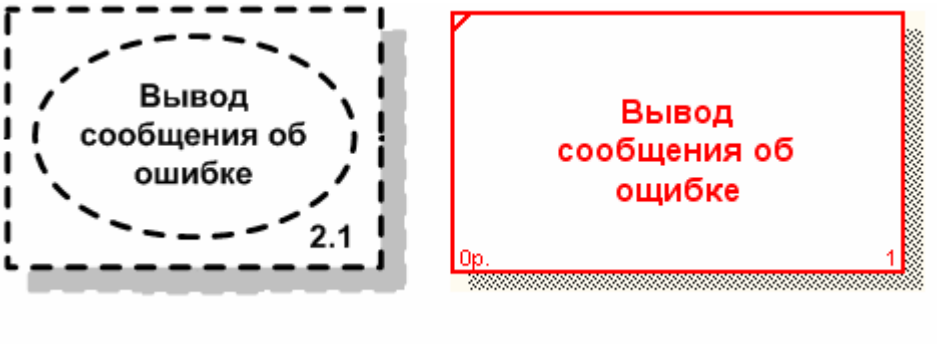

Рис. 7. Деятельность расширения

Деятельность расширения представляет собой вариант использования, представляющий один из шагов альтернативного потока варианта использования, описываемого диаграммой. Номер деятельности расширения, соответствует номеру шага альтернативного потока, на который он отображается. Поскольку шаги альтернативного потока варианта использования нумеруются в зависимости от номера шага нормального потока, на котором происходит переход на альтернативный поток варианта использования, то средствами стандартных инструментов SADT-моделирования, невозможно добиться требуемой нумерации работ.

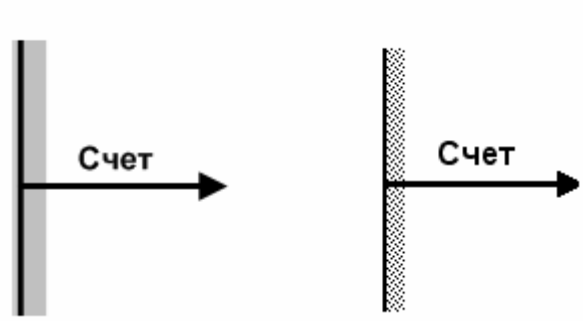

Рис. 8. Выход при успешном завершении работы

Выход при успешном завершении работы не только отражает цель варианта использования, выполняющегося на шаге потока, но и вносит в концепцию вариантов использования информационные связи между вариантами использования, что значительно повышает ценность этого подхода. Отсутствие информационных связей между элементами модели, описываемой вариантами использования, является одним из основных недостатков данного похода к моделированию функциональных требований. Количество выходов каждой работы может быть любым, при этом каждый выход отражает возможность перехода к разным шагам основного потока варианта использования, после завершения шага, на который отображается данная работа. Однако при моделировании вариантов использования с помощью SADT-диаграмм необходимо избегать использования абстрактных работ, которые объединяют несколько работ, по

какому-либо признаку. Примером такого объединения может служить работа «Управлять выполнением задания» из прошлой главы, которая отображается на три варианта использования: «Оценить степень завершенности задания», «Принять готовую деталь» и «Разработать план выполнения задания».

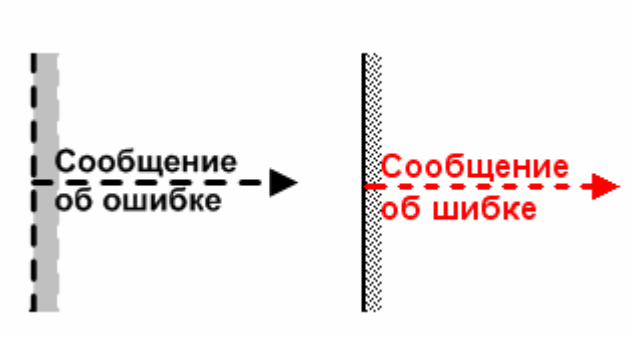

Рис. 9. Выход при неуспешном завершении работы

Выход при неуспешном завершении работы связывает шаг варианта использования с расширением, возможным на этом шаге, кроме того, он устанавливает информационную связь между шагом основного потока варианта использования и его расширением. Такой подход позволяет не декомпозировать расширения варианта использования, поскольку во многих случаях семантика действий, происходящих в расширении, становится ясна из его информационных связей и названия. Выход при неуспешном завершении работы не является целью варианта использования, однако может выполнять минимальные гарантии варианта использования.

Рис. 10. Вход работы основного потока

Входы работы основного потока являются предусловиями, которые должны быть выполнены для начала выполнения варианта использования. Триггером варианта использования, отображающимся на данную работу, является либо наличие всех ресурсов, которые описываются входами работы, либо событие, которое указывается автором диаграммы в описании данной работы. Однако в большинстве случаев для описания триггера варианта использования достаточно появления всех ресурсов, описываемых входами работы, в наличие.

Рис. 11. Вход работы расширения

Входы работы основного потока являются предусловиями, которые должны быть выполнены для начала выполнения расширения варианта использования. Предусловия для расширения варианта использования могут объединяться по принципу логического «И» или по принципу логического «ИЛИ», однако средствами стандартного инструмента моделирования SADT-диаграмм невозможно добиться визуализации такого объединения. В стандартном инструменте моделирования объединение входов достигается указанием на объединение в описании входа.

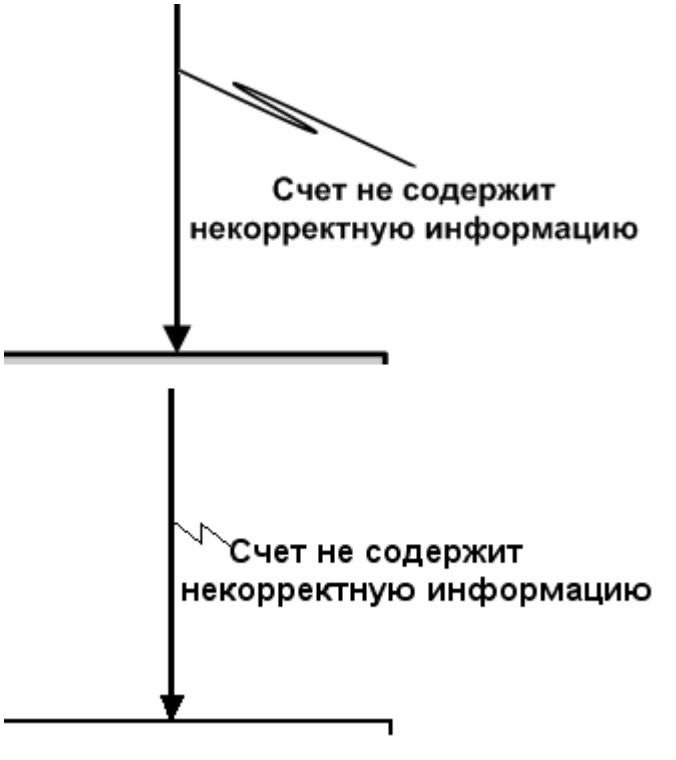

рис.12. Гарантии

Управление работы является минимальной гарантией для данного варианта использования, то есть условием, которое должно выполняться при любом ходе развития варианта использования.

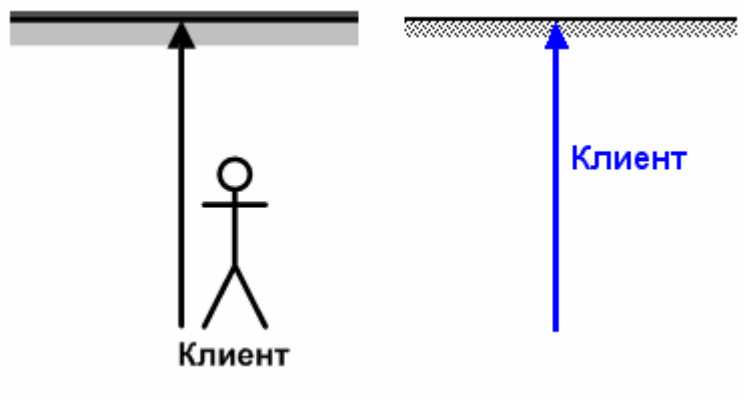

Рис. 13. Действующее лицо

Механизм работы может исполнять две функции при описании варианта использования с помощью модифицированной SADT-диаграммы. Первая функция механизма работы является обозначение действующих лиц варианта использования. В классических SADT-диаграммах механизм может также обозначать вспомогательные ресурсы, необходимые для выполнения работы, однако для описания вариантов использования необходимо отказаться от употребления данной функции механизма.

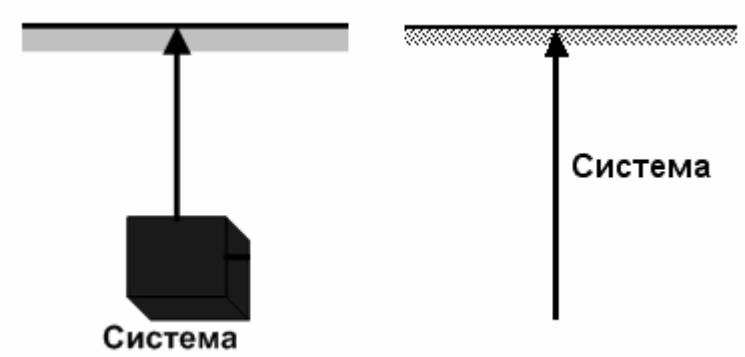

Рис. 14. Моделируемая система или ее часть

Вторая функция механизма в модифицированных SADT-диаграммах - это обозначение системы или части системы, которая участвует в выполнении данного варианта использования. Система, участвующая в выполнении варианта использования, также является действующим лицом этого варианта использования, однако явное указание системы или ее части необходимо для упрощения выделения подсистем проекта и разделения ответственности за реализацию функциональности этих подсистем.

Все элементы предлагаемой нотации представляют собой специализацию элементов SADT-диаграмм, что делает нотацию легкой для освоения проектировщиками, использующими SADT-диаграммы для описания функциональных требований к системе с одной стороны. С другой стороны, каждая диаграмма представляет собой графическое описание варианта использования, использующее привычную для проектировщиков вариантов использования терминологию и несущее практически ту же смысловую нагрузку, что и текстовое описание. Во многих случаях семантическая значимость модифицированных SADT-диаграмм, описывающих вариант использования, оказывается выше, чем стандартное текстовое описание, благодаря внесению явного указания на информационные и иерархические связи между вариантами использования.

Пример варианта использования, записанный в предлагаемой нотации:

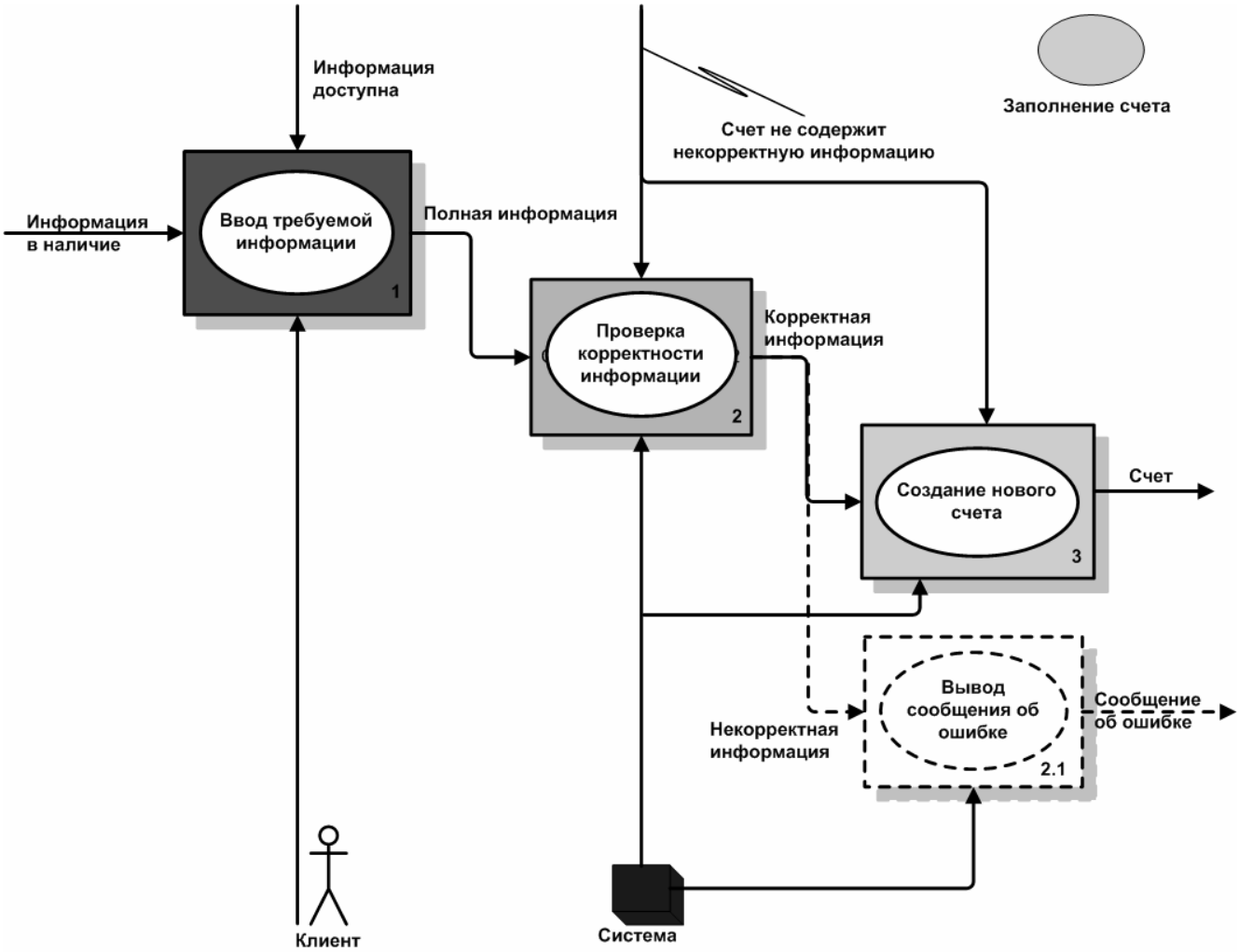

Рис. 15 Вариант использования «Заполнение счета»

Представленная диаграмма является графическим представлением варианта использования «Заполнение счета». Каждый элемент «работа», вне зависимости от того, какому потоку он принадлежит - основному или альтернативному, является самостоятельным вариантом использования и может быть декомпозирован с помощью аналогичных диаграмм. Этот же вариант использования может быть описан с помощью диаграммы, полученной с помощью стандартного инструмента SADT-моделирования, однако использование такого инструмента является временным решением.

При практическом применении такого подхода к описанию вариантов использования была отмечена простота, с которой создаются и читаются варианты использования. Однако использование стандартных инструментов SADT-моделирования, вызывало многочисленные трудности, поскольку многие возможности, заложенные в этом подходе, либо невозможно реализовать с помощью стандартного инструмента моделирования, либо их реализация вызывает затруднения у аналитика. Опыт аналитиков, которые применяли этот метод для описания функциональных требований к системе, используя при этом стандартные средства моделирования SADT-диаграмм, послужил основным толчком для начала создания инструмента SADT-моделирования, которые бы позволял добавлять расширения в графический язык проектирования. Возможность

расширения этого языка без внесения изменений в код ядра инструмента стала основным требованием к архитектуре будущего инструмента.

3. Функциональные требования к расширяемому ядру инструмента SADT моделирования

Функциональные требования к расширяемому ядру инструмента SADT моделирования описаны с помощью классических вариантов использования, которые объединены в диаграммы вариантов использования [11]. В рассматриваемой системе существует только два действующих лица – «Пользователь», который строит SADT диаграммы с помощью инструмента и «Программист», который добавляет расширения в ядро инструмента или изменяет его поведение. Совокупность функциональных требований, описанных с помощью вариантов использования, для пользователя и для программиста составляет полный набор функциональных требований к расширяемому ядру инструмента SADT моделирования. В настоящий момент не выявлено жестких нефункциональных требований и ограничений, поскольку графическая подсистема выбранной платформы, а именно Java 1.5, не накладывает ограничений на производительность системы.

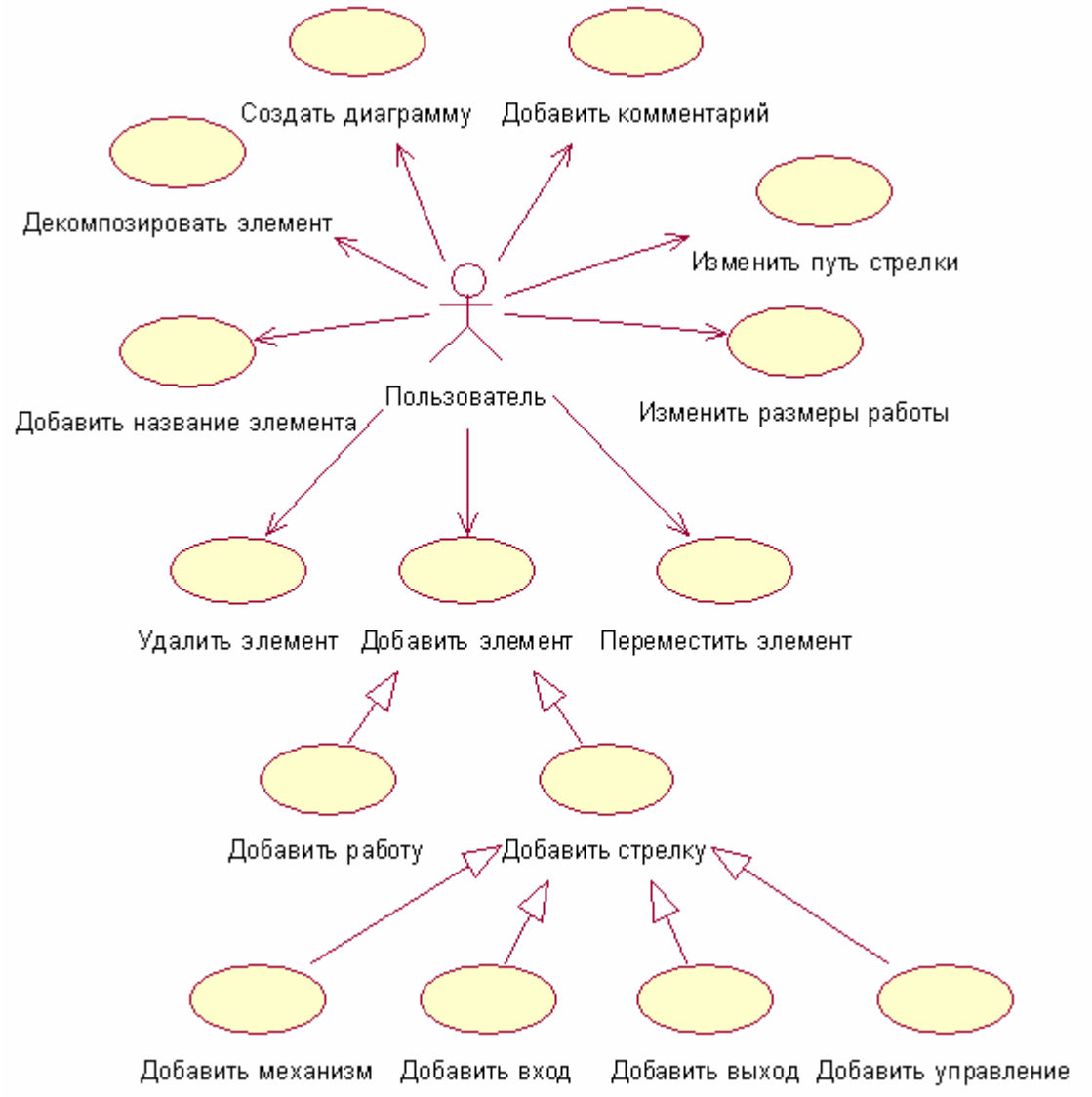

Рис. 16. Варианты использования для действующего лица «Пользователь»

Варианты использования, которые относятся к пользователю, мало отличаются от вариантов использования, которые разрабатывались бы для создания классического инструмента SADT моделирования. Эти варианты использования описывают взаимодействие пользователя с диаграммой, которая является графическим представлением моделируемого функционального требования.

Напротив, варианты использования, относящиеся к действующему лицу «Программист», весьма специфичны, поскольку именно они реализуют основную особенность данной системы. Соответственно, именно варианты использования, относящиеся к действующему лицу «Программист», являются архитектурно значимыми, и именно они определили выбор платформы и использованные паттерны проектирования.

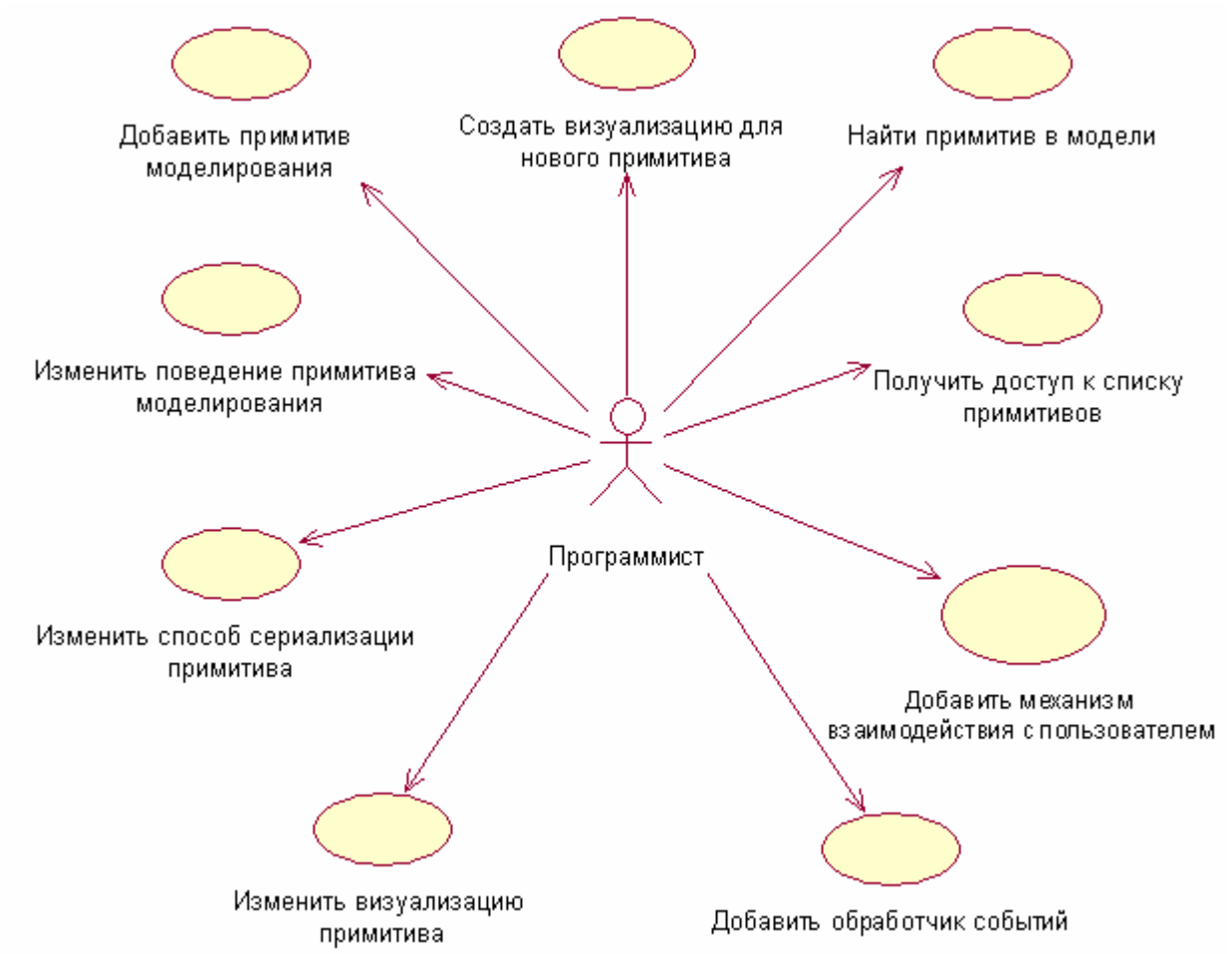

Рис. 17. Варианты использования для действующего лица «Программист»

4. Архитектура расширяемого ядра инструмента SADT-моделирования

4.1 Архитектура уровня управления моделью

Модуль, отвечающий за иерархию и хранение модели, включает в себя ряд классов, которые всегда присутствуют в системе и используются в коде, управляющем графическим интерфейсом пользователя и другими внешними модулями. К ним относятся классы Author, представляющий автора проекта, класс Project, представляющий проект, класс Model, представляющий модель функциональных требований, класс Diagram, представляющий отдельную диаграмму и класс ModelTree, который организует древесную иерархию диаграмм.

В классе, представляющем автора проекта, хранится имя и инициалы автора проекта. Кроме того, через объект этого класса можно получить доступ к коллекции всех проектов, принадлежащих этому автору. В отличие от остальных предопределенных классов модуля, этот класс не имеет менеджера, и его экземпляры создаются с помощью фабричного метода getInstance(), принадлежащего самому классу Author. Описание всех авторов, которые создавали проекты с помощью инструмента, хранится в отдельном XML файле authors.xml. В этом же файле находится ссылка на алгоритм, который использовался для получения хэша из пароля автора.

```
<?xml version="1.0" encoding="UTF-8" ?>
<list>
  <digestMethod algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
  <author>
    <name>
      Oleg A. Zmeev
    </name>
    <initials>
      O. Z.
    </initials>
    <password>
      a0b4luuk...
    </password>
  </author>
  <author>
    <name>
      Anton U. Malinovskiy
    </name>
    <initials>
      A. M.
    </initials>
    <password>
      a9qqqIrt...
    </password>
  </author>
</list>
```

Рис. 18. Пример XML файла описания авторов

Класс Project является самым верхним классом в иерархии моделирования. Кроме названия, описания и автора проекта из объекта этого класса можно получить коллекцию всех моделей, которые в него входят. Каждый проект используется для описания функциональных требований одной или нескольких систем, объединенных общим назначением. Для описания проекта формируется собственный XML-файл, в котором хранится информация обо всех моделях, входящих в проект.

```
<?xml version="1.0" encoding="UTF-8" ?>
<project title="Sample Projet" author="Anton U. Malinovskiy">
  <model title="Sample1">
    <type>
      IDEF0
    </type>
    <format>
      IDL
    </format>
    <path>
      Samp11/sample1.idl
    </path>
  </model>
  <model title="Sample2">
    <type>
      ModifiedSADT
    </type>
    <format>
      xml
    </format>
    <path>
      Samp12/sample2.xml
    </path>
  </model>
</project>
```

Рис. 19. Пример XML файла проекта

Каждая система, для которой моделируются функциональные требования, представляется отдельным объектом класса Model, который содержится в коллекции моделей проекта. Функциональные требования к системе моделируются с помощью отдельных диаграмм, которые организованы в древесную иерархию. Каждый уровень иерархии является декомпозицией элементов предыдущего уровня.

Отдельные диаграммы представляются экземплярами класса Diagram. Уникальными идентификаторами всех преопределенных классов, кроме класса представляющего диаграмму, образующих иерархию моделирования являются их названия (Title). Уникальным идентификатором объекта диаграммы, согласно стандарту IDEF0 является ее номер (например, номер A11 указывает на диаграмму, которая находится на третьем уровне иерархии и является декомпозицией первого элемента на диаграмме A1). Получение объектов из коллекции, которая принадлежит объекту, находящемуся выше по иерархии происходит с помощью Get методов, в которые передается название этого объекта или номера для диаграммы.

Для организации древесной иерархии класс ModelTree реализует интерфейс TreeModel из стандартного пакета javax.swing.tree, класс Diagram реализует интерфейс TreeNode из того же пакета. Работать с древесной структурой диаграмм можно как с помощью методов, предоставляемых менеджером объекта модели, так и напрямую с объектом древесной структуры, предварительно получив его из объекта модели. Так, например, получить отдельную диаграмму из структуры можно как, воспользовавшись методом getDiagram(), которому передается название необходимой диаграммы, так и с помощью перебора всей древесной структуры.

Поскольку класс древесной структуры диаграмм является реализацией стандартного интерфейса древесной структуры пакета javax.swing, то для отображения древесной структуры целесообразно реализовать класс, наследующий от стандартного класса отображения древесной структуры JTree , чтобы использовать особенности классов DiagramTreeModel и Diagram.

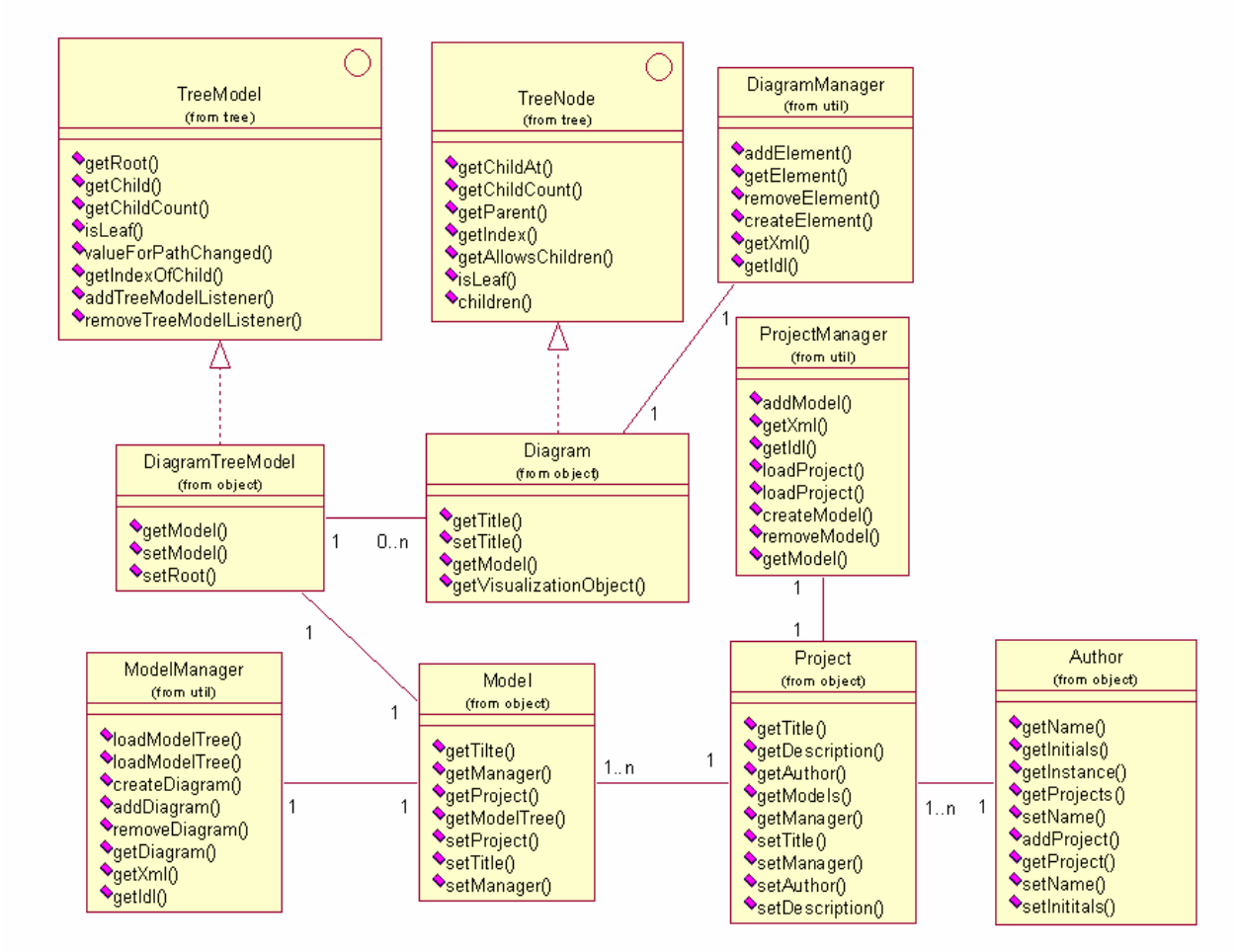

Рис. 20. Предопределенные классы модуля управления моделью

Для работы с этими классами используются управляющие объекты, получить которые можно через метод getManager(). Такой подход позволяет не только разгрузить код соответствующих классов и сделать его проще для понимания и модификации (использование этого подхода эквивалентно применению рефакторинга «Выделение класса»), но и добавляет системе необходимую гибкость. Этим объектам делегируется большая часть управляющей функциональности, что позволяет модифицировать поведение класса, подменяя базовую реализацию менеджера собственной. Эта реализация

должна наследовать базовую и перегружать методы, которые обеспечивают требуемую модификацию поведения объекта.

Сами предопределенные классы являются сущностями аналитической модели и инкапсулируют информацию, имеющую значение для конечного пользователя. Они, в отличие от объектов менеджеров, имеют состояние, которое отражает текущее состояние моделирования, и, соответственно, могут быть сериализованы и десериализованы для обеспечения возможности хранения их состояния. В настоящее время предусмотрена возможность сериализации в XML и IDL представление.

Проект должен содержать модели, созданные в одной нотации, которая описывается полем типа ProjectType объекта проекта (в XML представлении тип проекта описывается в теге <type>). Однако модели, которые входят в проект могут храниться как в XML, так и в IDL представлении. Это позволяет использовать для редактирования некоторых моделей проекта сторонние инструменты, а также импортировать модели созданные в этих инструментах в проект.

Хранение состояния в виде XML представления позволяет легко добавлять расширения к существующей нотации, однако модели, хранимые в этом представлении, не могут использоваться сторонними инструментами.

```
<?xml version="1.0" encoding="UTF-8" ?>
<model title="PhotositePortal" project="PhotositeProject">
  <creationDate>
    16/2/2005
  </creationDate>
  <status>
    working
  </status>
  <timeFrame>
    AS-IS
  </timeFrame>
  <pointOfView>
    User
  </pointOfView>
  <description>
    User interaction with the PhotoSite portal
  </description>
  <diagramStructure>
    <diagram id="A0" title="PhotoSite interation" parent="" element="">
      <creationDate>
        16/2/2005
      </creationDate>
      <revisionDate>
        17/2/2005
      </revisionDate>
      <elementList>
        ...
      </elementList>
    </diagram>
    <diagram ...
  </diagramStructure>
</model>
```

Рис. 21. Пример XML файла модели

Кроме XML представления, которое необходимо в основном для хранения моделей, созданных в расширенной нотации. Состояние модели может сохраняться в формате IDEF IDL, который предназначен для хранения IDEF0 диаграмм. Хотя возможности, которые предоставляет архитектура ядра инструмента моделирования, позволяют расширять IDL представление модели, но для хранения расширенных моделей лучше воспользоваться XML представлением.

```
KIT ;
  IDL VERSION 1.2.8 ;
  TITLE 'User interation with PhotoSite' ;
  AUTHOR 'Anton U. Malinowski' ;
  CREATION DATE 16/3/2005 ;
  PROJECT NAME 'PhotoSiteProject' ;

  MODEL 'Fotopark.ru'

    AUTHOR 'Anton U. Malinowski' ;
    PROJECT NAME 'PhotoSiteProject' ;

    DIAGRAM GRAPHIC A-0 ;
      CREATION DATE 16/3/2005 ;
      REVISION DATE 17/3/2005 ;
      TITLE 'User interation with PhotoSite' ;
      STATUS WORKING ;

      BOX 0 ;
        NAME 'User interation' ;
        ...
```

Рис. 22. Пример IDL файла модели

Изменение поведения предопределенных классов возможно через загрузку собственных менеджеров, которые реализуют паттерн «Фабрика» [12] для соответствующих классов. Кроме создания экземпляров предопределенных классов, менеджеры отвечают за сохранение состояния управляемых объектов в XML и IDL представление. Менеджеры с измененным поведением должны быть потомками соответствующих классов и описываться в конфигурационном файле инструмента моделирования.

Базовая реализация сохранения состояния в XML файлах использует технологию JAXB, которая позволяет отображение XML файлов на структуру Java объектов. Такой подход позволяет легко расширять существующую нотацию, создавая собственные описания XML нотаций в XSD схемах. Кроме того, производительность работы с XML файлами, для которых создано отображение на Java классы, намного выше, чем при использовании DOM анализаторов, которым необходимо держать в памяти всю древесную структуру XML файла. После модификации XSD схемы необходимо воспользоваться инструментом, поставляемым компанией Sun вместе с пакетом JAXB, для генерации классов, являющихся отображением XML файла.

```
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
 <xs:element name='creationDate' type='xs:date'>
 </xs:element>
 <xs:element name='description' type='xs:string'>
 </xs:element>
 <xs:element name='diagram'>
  <xs:complexType>
   <xs:sequence>
     <xs:element ref='creationDate'/>
     <xs:element ref='revisionDate'/>
     <xs:element ref='description'/>
     <xs:element ref='elementList'/>
   </xs:sequence>
  <xs:attribute name='element' use='required' type='xs:IDREF'/>
  <xs:attribute name='parent' use='required' type='xs:IDREF'/>
  <xs:attribute name='title' use='required'/>
  <xs:attribute name='id' use='required' type='xs:ID'/>
  </xs:complexType>
 </xs:element>
 <xs:element name='diagramStructure'>
  <xs:complexType>
   <xs:sequence>
     <xs:element ref='diagram'/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name='element'>
 ...
```

Рис. 23. Пример XSD описания модели

Реализация сохранения состояния в IDL файлах использует встроенный грамматический анализатор, созданный для разбора языков, грамматика которых описывается также с помощью XML файлов.

Кроме преопределенных классов модуль управления моделью включает в себя ряд классов, представляющих примитивы SADT моделирования. Классы, представляющие собой примитивы моделирования, должны реализовывать интерфейс ModelElement. Они также описываются в конфигурационном файле инструмента моделирования.

В ядро инструмента проектирования включен набор базовых классов, который реализует примитивы SADT проектирования: Работа (класс Action), Вход (класс Input), Выход (класс Output), Управление (класс Control), Механизм (класс Mechanism). Они также реализуют интерфейс ModelElement и описываются в конфигурационном файле. Это дает возможность изменить поведение этих классов, создав собственный класс, являющийся потомком одного из базовых и перегрузив соответствующие методы класса, либо создать собственную реализацию интерфейса ModelElement.

Каждый объект, представляющий примитив моделирования, сам отвечает за сохранение своего состояния в XML или IDL виде. При этом информация о графической составляющей модели берется из соответствующих объектов визуализации.

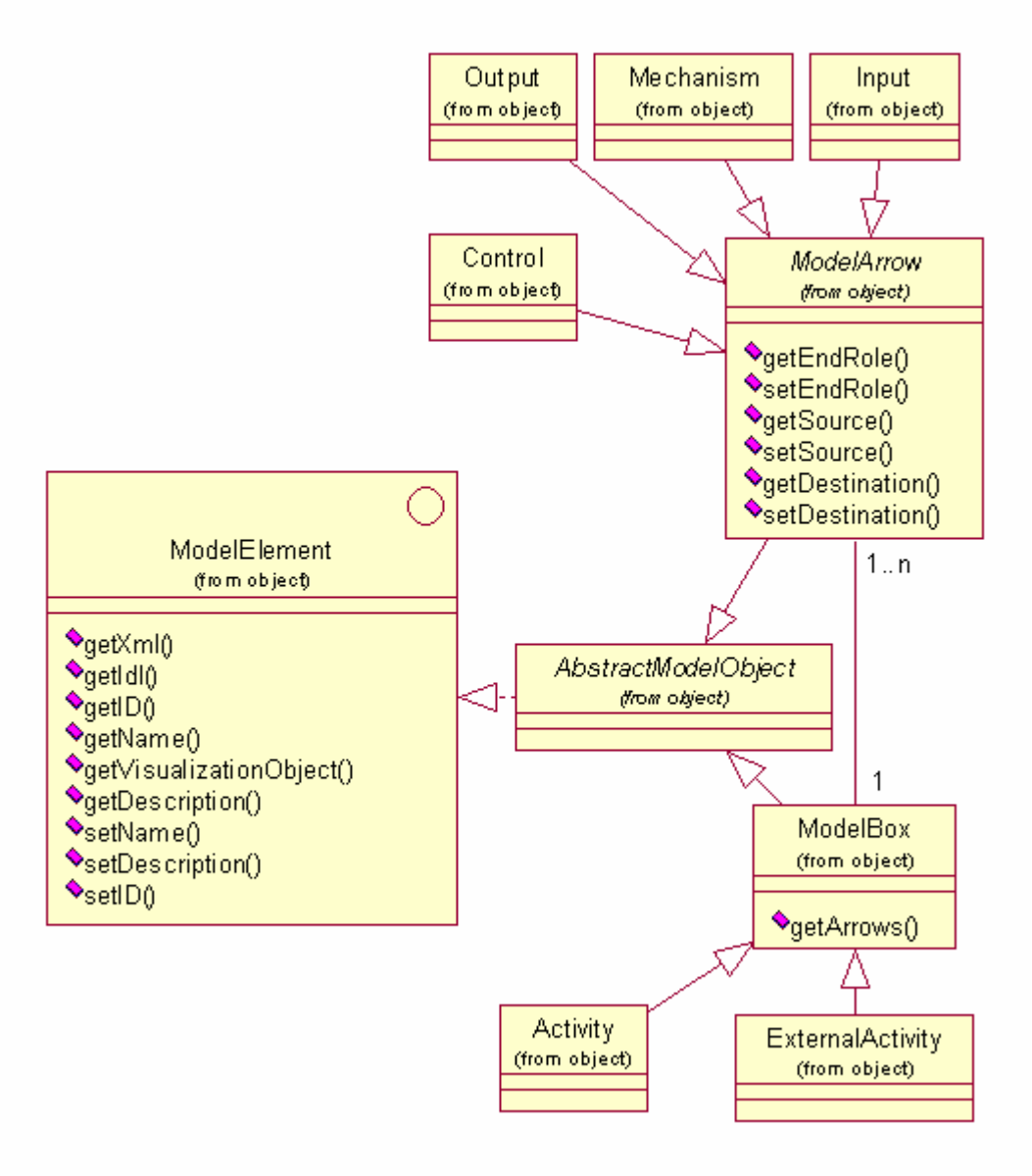

Рис.24. Классы SADT примитивов уровня управления моделью

Доступ к объектам примитивов моделирования, которые содержатся в диаграмме, осуществляется через поиск по уникальному идентификатору, который должен иметь каждый объект диаграммы. Поиск объектов осуществляется либо посредством навигации по связям между объектами, либо с помощью поиска по уникальному идентификатору, который предоставляется объектом менеджера диаграммы.

Модуль уровня управления моделью обеспечивает целостность модели и сохранение ее состояния в XML и IDL форматах, кроме того, архитектура этого модуля позволяет создавать расширения существующих нотаций и использовать модели, полученные в расширенных нотациях, в сторонних инструментах анализа и проектирования.

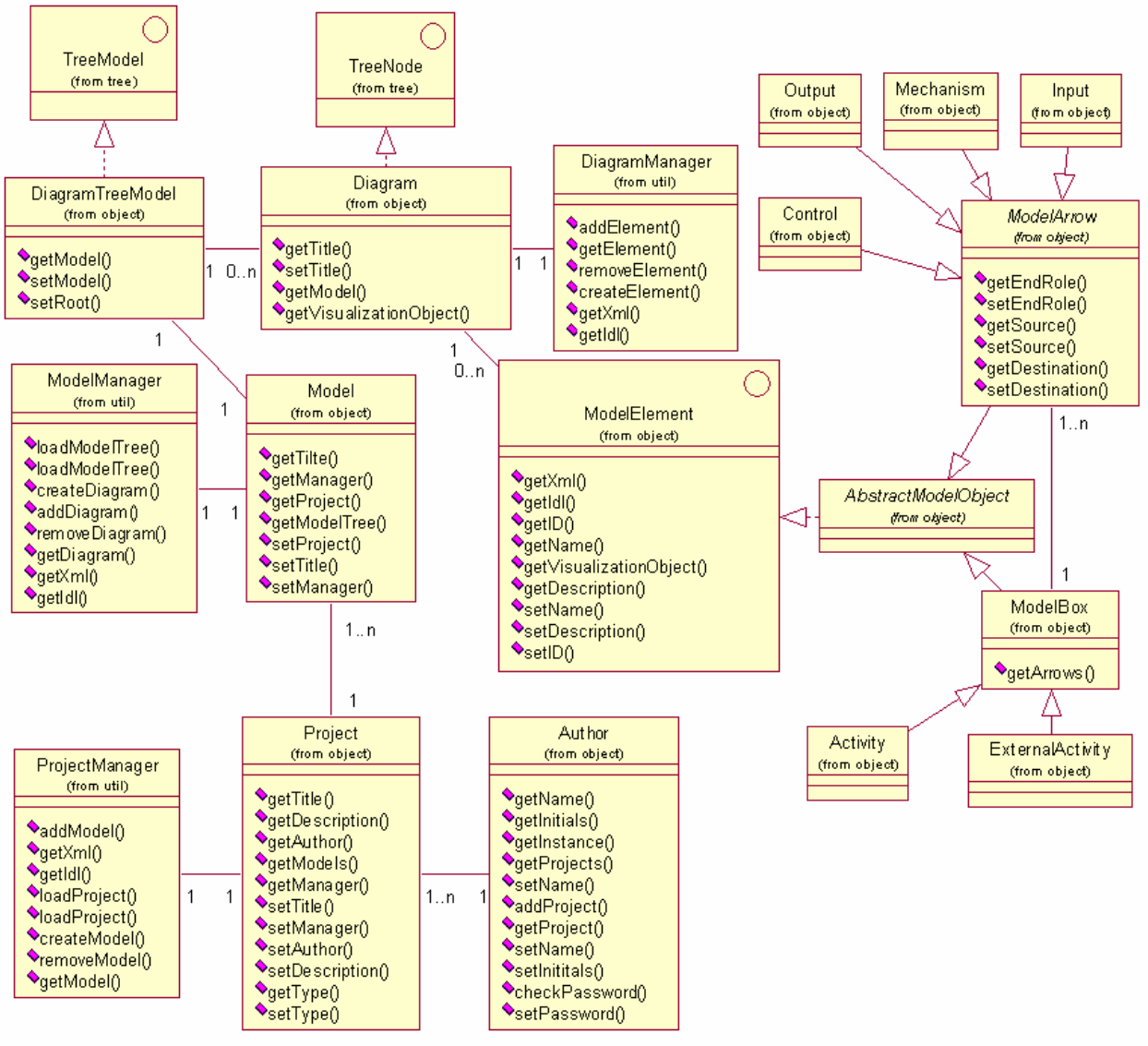

Рис.25. Архитектура уровня управления моделью

Гибкость и открытость архитектуры достигается за счет использования XML ориентированных технологий, таких как JAXB и XmlSchema. С одной стороны эти технологии предоставляют возможности для типизации XML файлов, что позволяет использовать модели, созданные с помощью инструмента, для дальнейшего проектирования. С другой - возможности расширения и изменения нотаций моделей.

Использование подхода к проектированию, основывающемуся на объектах-менеджерах, которые инкапсулируют такие задачи как сохранение состояния объектов, их десериализации, создание экземпляров и прочие функциональные задачи, позволяет изменять функциональность инструмента для соответствия измененной нотации моделирования. Благодаря изолированности объектов-менеджеров возможно локальное изменение функциональности ядра инструмента моделирования, которое не затрагивает других частей ядра или других модулей системы. Взаимодействие модулей системы с ядром также происходит через объекты-модули, что позволяет, при необходимости, изменить или расширить интерфейс взаимодействия, не внося изменения в код предопределенных модулей системы.

Кроме того, этот подход упрощает последующую модификацию кода модуля, скрывая подробности реализации низкоуровневых задач в коде объектов-менеджеров. В

то же время базовой реализации преопределенных классов, классов, представляющих примитивы моделирования, и объектов-менеджеров достаточно для реализации полноценного инструмента SADT моделирования.

4.2 Архитектура уровня отрисовки модели

Архитектура уровня отрисовки модели также делится на две части: преопределенную, которую нельзя заменить собственной без изменения исходного кода системы, и конфигурируемую часть, которая определяется в конфигурационном файле инструмента. В отличие от модуля, управляющего моделью системы, модуль, отвечающий за отрисовку модели, включает только один преопределенный класс DiagramPane, который представляет собой визуализацию диаграммы.

Он наследует стандартный класс панели JPanel из пакета swing и используется для построения графического интерфейса пользователя. Этот класс отвечает за отображение панели рисования диаграммы и является связующим звеном между уровнем графического интерфейса пользователя и ядром инструмента SADT моделирования. Все действия пользователя, которые имеют отношение к рисованию модели, обрабатываются менеджером этого объекта и делегируются далее по дереву обработки событий.

Отрисовка всей диаграммы осуществляется с помощью вызова метода draw() объекта панели диаграммы и непосредственно выполняется специальным объектом класса DiagramRenderer, который инкапсулирует методы, имеющие отношение непосредственно к рисованию графических примитивов. Для внесения изменений, касающихся отображения графических примитивов диаграммы, необходимо создать собственный класс, наследующий от класса DiagramRenderer. Этот класс также описывается в конфигурационном файле инструмента.

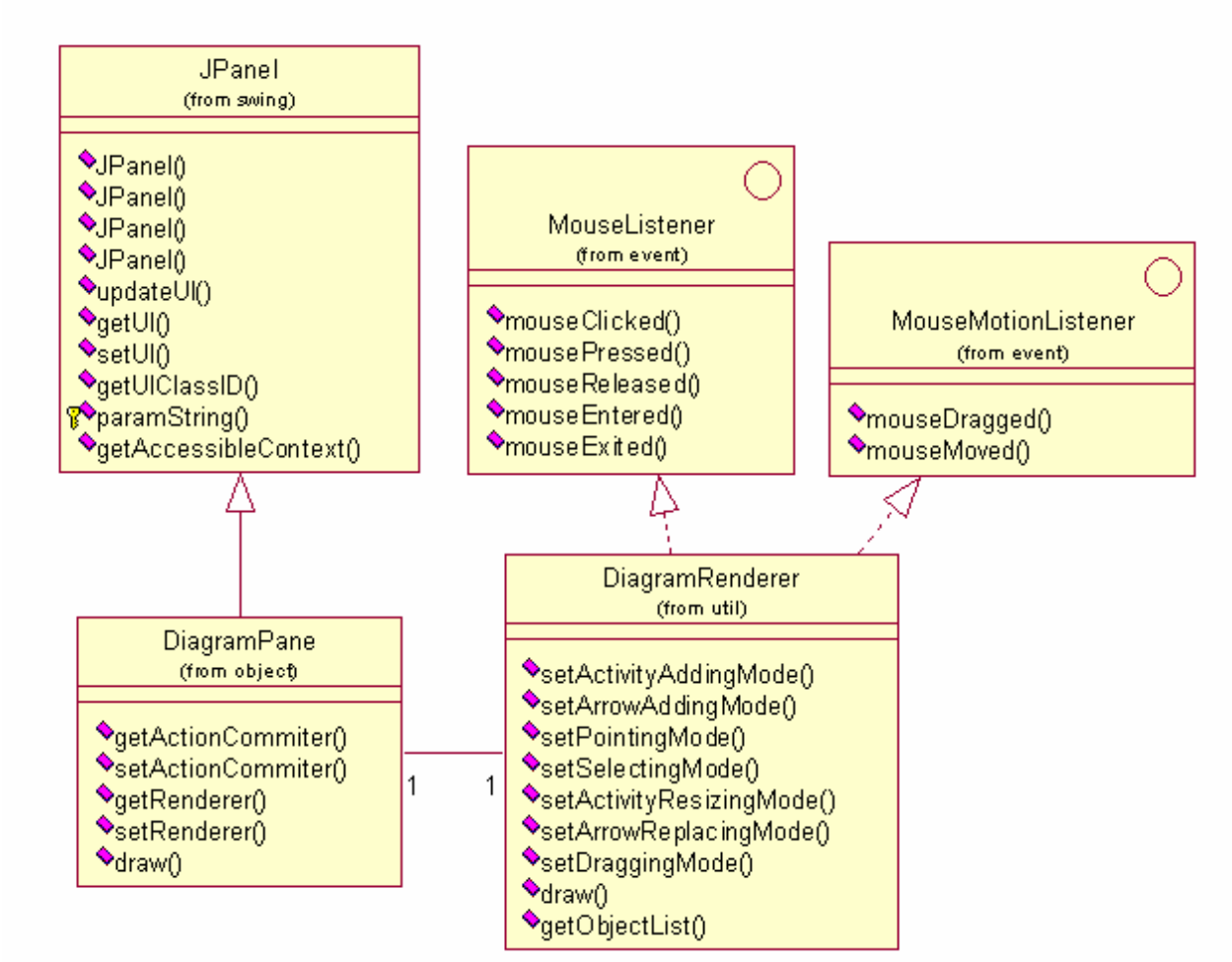

Рис.26. Архитектура панели диаграммы

Объект класса DiagramRenderer также получает все события, которые порождаются действиями пользователя, поскольку реализует интерфейсы различных «подписчиков» [6] для событий объекта класса DiagramPane. Этот объект не реализует логику взаимодействия с пользователем, делегируя обработку события далее.

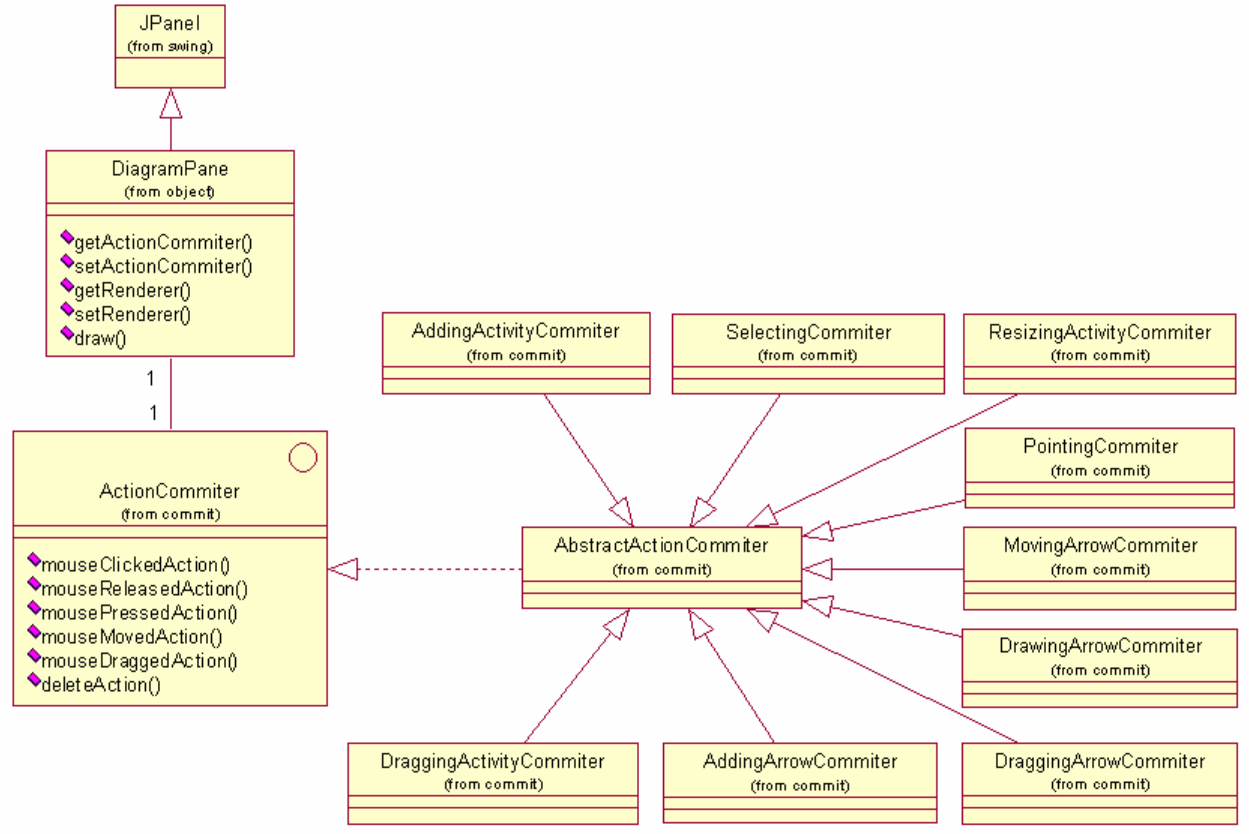

Рис.27. Классы, реализующие взаимодействие с пользователем

Взаимодействие пользователя с диаграммой происходит посредством набора классов, реализующих интерфейс ActionCommiter и представляющих собой реализацию паттерна «Стратегия» [12]. Переключение текущего объекта класса, реализующего интерфейс ActionCommiter, осуществляется объектом, класса DiagramRenderer, что позволяет добавлять новые режимы взаимодействия пользователя с диаграммой или изменить базовые.

Классы, реализующие взаимодействие с пользователем:

- AddingActivityCommiter – реализует добавление новой деятельности на диаграмму.
- AddingActivityCommiter – реализует добавление нового объекта типа стрелка на диаграмму.
- SelectingCommiter – реализует выбор объектов на диаграмме.
- ResizingActivityCommiter – реализует изменение размеров деятельности.
- PointingCommiter – обрабатывает перемещение курсора по диаграмме, в этот момент никакого действия пользователь не совершает.
- MovingArrowCommiter – реализует перемещение отельных участков объекта типа стрелка

- DrawingArrowCommiter – реализует создание объекта типа стрелка на диаграмме
- DraggingArrowCommiter – реализует перетаскивание объекта типа стрелка на диаграмме
- DraggingAcitvityCommiter – реализует перетаскивание деятельности на диаграмме

Вторая часть уровня отрисовки модели состоит из объектов, визуализирующих элементы диаграммы. Все элементы диаграммы являются классами, реализующими интерфейс DrawingObject, их экземпляры создаются с помощью метода GetVisualizationObject соответствующего объекта модели.

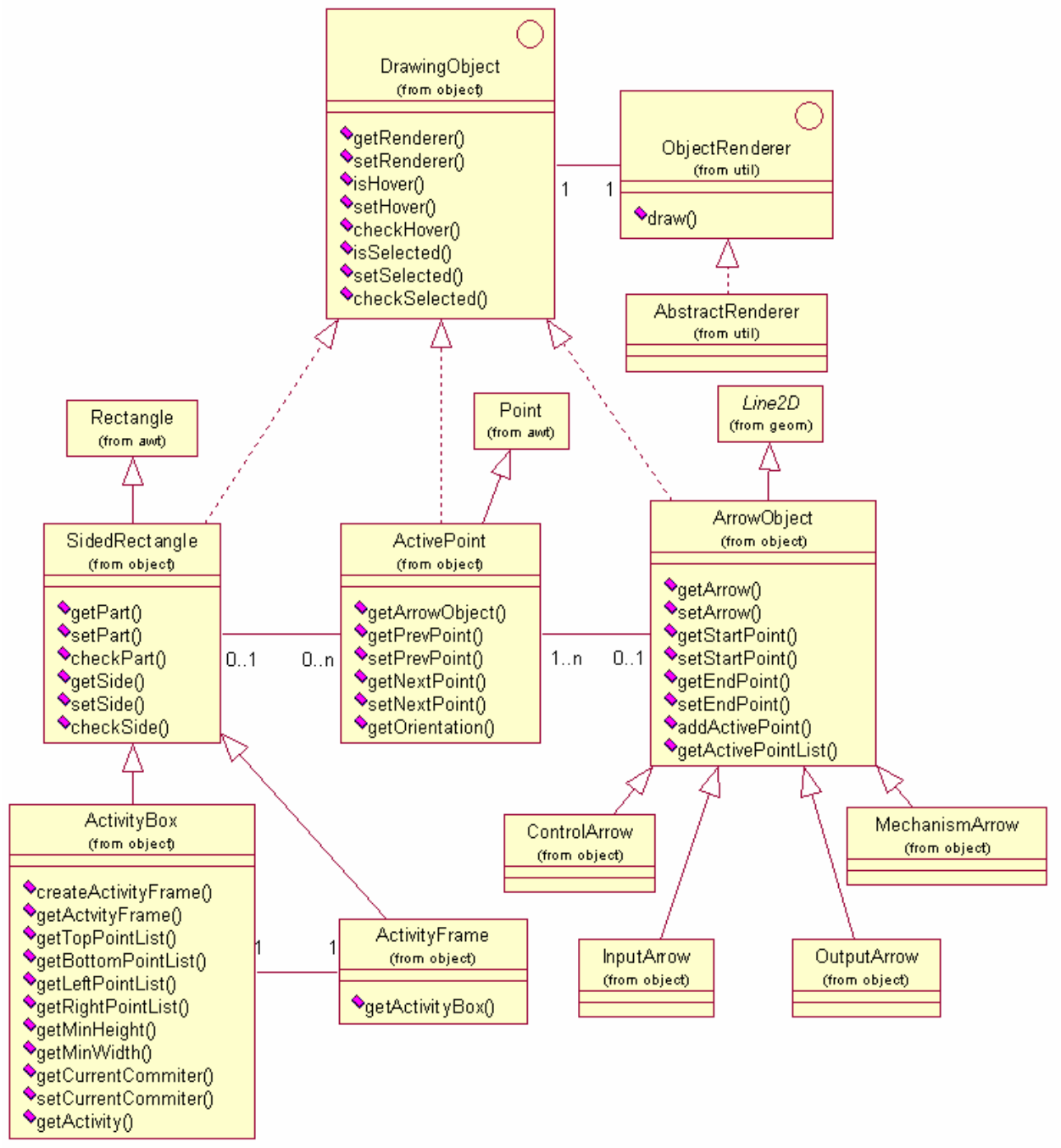

Рис.28. Классы, визуализирующие элементы диаграмм

Соответственно после реализации собственного визуализирующего объекта необходимо создать новую реализацию класса, соответствующего объекту модели, с перегруженным методом GetVisualizationObject.

Также для реализации собственного примитива моделирования необходимо либо создать класс, реализующий интерфейс ObjectRenderer, который будет осуществлять отрисовку соответствующего элемента, либо унаследовать его от одной из базовых реализаций этого интерфейса и перегрузить соответствующие методы.

В базовую реализацию ядра инструмента SADT моделирования входят следующие классы, реализующие визуализацию объектов диаграммы.

- SidedRectangle – базовый класс для всех прямоугольных форм на диаграмме, (в базовой реализации это ActivityBox и ActivityFrame). Этот класс является наследником стандартного класса прямоугольника из пакета java.awt (Rectangle) и перегружает методы, отвечающие за изменение размеров прямоугольника и определение относительного расстояния между прямоугольниками. Кроме того, инициирует обработку событий при изменении положения прямоугольника и изменении его размеров у объектов – подписчиков.
- ActivePoint – активная точка на диаграмме. Может находиться в месте изменения направления рисования объекта, визуализирующего стрелку, в месте сочленения объектов типа стрелка и в месте выхода стрелок из объектов, визуализирующих деятельность, и их входа в эти объекты. Организованы в двусвязный список для ускорения отрисовки диаграммы.
- ArrowObject – базовый класс для всех объектов визуализирующих стрелки на диаграмме (в базовой реализации к ним относятся InputArrowObject, OutputArrowObject, ControlArrowObject, MechanizmArrowObject). Содержит список активных точек, которые определяют путь для отрисовки объекта. Также предоставляет интерфейс для работы с этим списком. Для того чтобы найти деятельности, которые соединяет этот объект, необходимо воспользоваться соответствующими методами объекта уровня модели (в базовой реализации это объект класса Arrow).
- ActivityBox – класс, визуализирующий объекты типа деятельность на диаграмме. Является наследником класса SidedRectangle. Каждый объект этого класса содержит 4 списка активных точек для каждой из своих сторон и предоставляет интерфейс для работы с этими списками.
- ActivityFrame – класс, использующийся для отображения рамки деятельности, во время некоторых действий (например перемещение деятельности на диаграмме). Является наследником класса SidedRectangle и создается с помощью фабричного метода объекта класса ActivityBox.
- InputArrowObject - класс, визуализирующий объекты типа вход. Является наследником ArrowObject.
- OutputArrowObject - класс, визуализирующий объекты типа выход. Является наследником ArrowObject.
- ControlArrowObject - класс, визуализирующий объекты типа управление. Является наследником ArrowObject.
- MechanizmArrowObject - класс, визуализирующий объекты типа механизм. Является наследником ArrowObject.

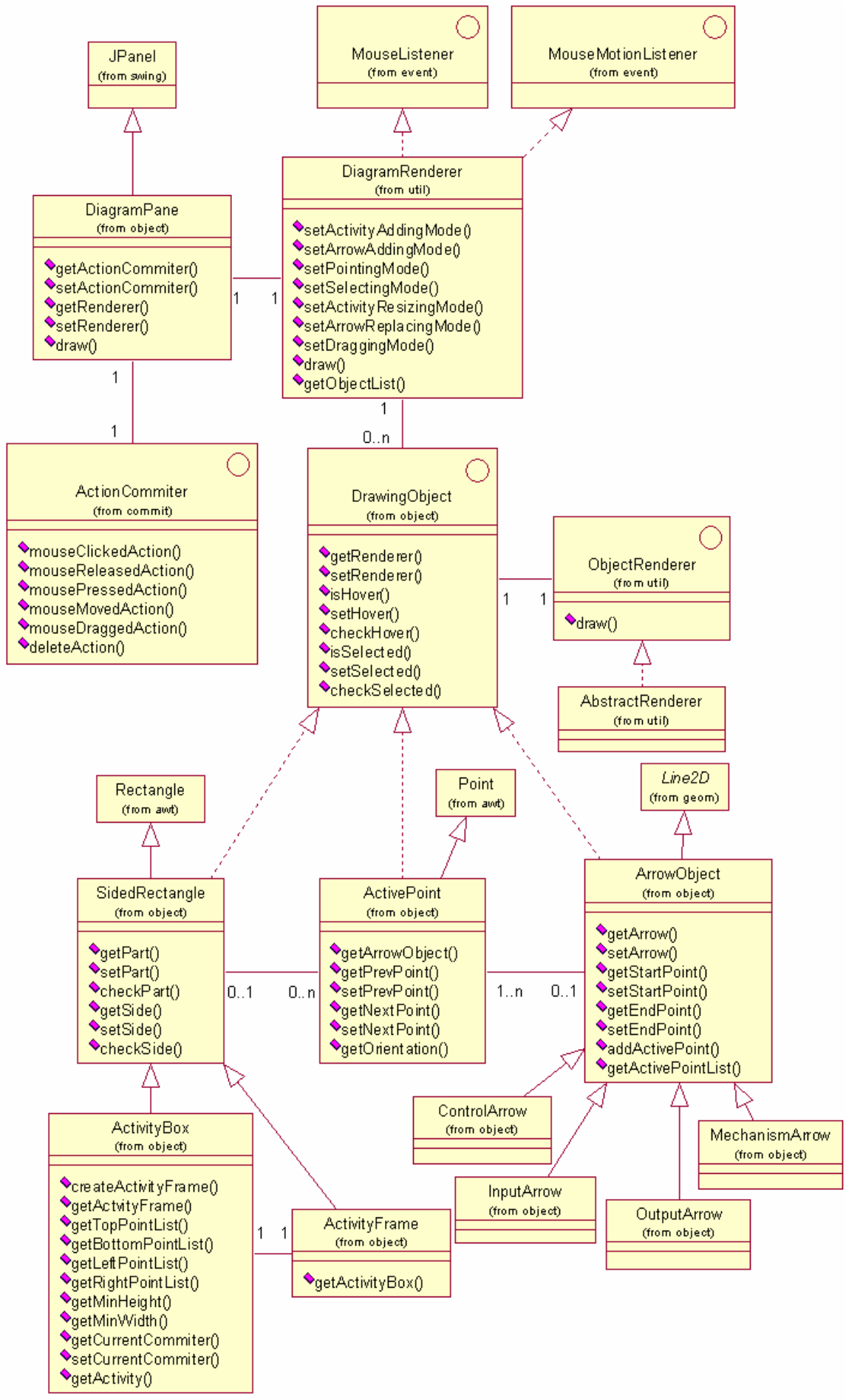

Рис.29. Классы, визуализирующие элементы диаграмм

Заключение

В данной работе был разработан метод описания функциональных требований к программному обеспечению, который объединяет подход, основанный на методологии структурного анализа и проектирования SADT, и подход, основанный на вариантах использования.

Была предложена нотация, рассчитанная на создание расширяемого инструмента SADT-моделирования, которая позволяет описывать варианты использования с помощью модифицированных SADT-диаграмм. Также предложена альтернативная нотация, которая позволяет использовать для частичной реализации данного подхода существующие инструменты SADT-моделирования.

Подход был внедрен для разработки функциональных требований к системам в компании «AllThingsCyber, inc.», занимающейся разработкой коммерческого программного обеспечения.

Были проанализированы функциональные требования к инструменту SADT моделирования, который сделает возможным полную реализацию предложенного метода. Была разработана архитектура расширяемого ядра SADT моделирования и создан прототип ядра инструмента, который реализует архитектурно значимые функциональные требования.

Список использованных источников

1. Марка Д.А., МакГоуэн К. Методология структурного анализа и проектирования SADT.- М.: 1993. – 191 с.
2. Фаулер М., Скотт К. UML в кратком изложении. Применение стандартного языка объектного моделирования: Пер. с англ. – М.:Мир, 1999. – 191 с.
3. Кратчет Ф. Введение в Rational Unified Process. 2-е изд.. : Пер. с англ. – М.: Вильямс, 2002. – 240 с.
4. Коберн А. Современные методы описания функциональных требований к системам. : Пер. с англ. – М.: Лори, 2002. – 263 с.
5. Якобсон А., Буч Г., Рамбо Дж. Унифицированный процесс разработки программного обеспечения. – СПб.: Питер, 2002. – 496 с.
6. Змеев О. А., Малиновский А. Ю. Автоматизация выделения вариантов использования из SADT-диаграмм --- // Теоретическая и прикладная информатика / Под ред. проф. А.Ф. Терпугова. – Томск: Изд-во Том.ун-та, 2004. – Вып. 1. – С. 18-26
7. National Institute of Standards and Technology. Integration Definition For Function Modeling (IDEF0). - Washington : Draft Federal Information, 1993.- 116.
8. Малиновский А. Ю. Специализация элементов SADT-диаграмм для записи вариантов использования --- // Материалы XLIII Международ. науч. студен. конф. «Студент и научно-технический прогресс: Информационные технологии». – Новосибирск: Новосиб. гос. ун-т, 2005. – С. 204 – 206
9. Eriksson H. Business modeling with UML.: New York: John Wiley & Sons, 1998.
10. Norman O. Business modeling UML vs. IDEF.: Griffith.: Griffith University, 2002. – 53 с.
11. Розенберг Д., Скотт К., Применение объектного моделирования с использованием UML и анализ прецедентов: Пер с англ. – М.: ДМК Пресс, 2002. – 160 с.
12. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного программирования. Паттерны проектирования. СПб.: Питер, 2004. – 386 с.

Приложение А. Руководство программиста

Выбор среды разработки

Для реализации прототипа системы была выбрана платформа Java. Версия JDK, которая использовалась для реализации прототипа 1.5, однако проект может быть также запущен в средах JRE 1.3.x – JRE 1.4.x. Для компиляции проекта можно использовать любую среду разработки, предназначенную для создания Java проектов. Автором была использована бесплатная интегрированная среда разработки Eclipse.

Фалы проекта:

- idef0modeller.java - основной класс проекта
- diagramPane.java – класс, реализующий панель диаграммы
- actionCommiter.java – интерфейс стратегии взаимодействия с пользователем
- abstractActonCommiter.java – базовый класс для стратегий взаимодействия
- addingActivityCommiter.java – класс стратегии добавления работы
- selectingCommiter.java – класс стратегии выбора элементов
- resizingActivityCommiter.java – класс стратегии изменения размеров работы
- pointingCommiter.java – класс стратегии ожидания
- movingArrowCommiter.java - класс стратегии перемещения концов стрелки
- drawingArrowCommiter.java – класс стратегии рисования стрелки
- draggingArrowCommiter.java – класс стратегии перемещения стрелки
- addingArrowCommiter.java – класс стратегии добавления стрелки
- draggingActivityCommiter – класс перемещения работы
- diagramRenderer.java – класс, реализующий отрисовку диаграммы
- drawingObject.java – интерфейс визуального объекта диаграммы
- objectRenderer.java – интерфейс класса, отрисовывающего элемент диаграммы
- abstractRenderer.java – базовый класс для, отрисовщиков элементов
- sidedRectangle.java – базовый класс для прямоугольных элементов диаграммы
- activePoint.java – класс активной точки дигаммы
- activePointRenderer.java – класс, отрисовывающий активную точку диаграммы
- arrowObject.java – базовый класс для стрелок на диаграмме
- outputObject.java – класс для элемента «выход» на диаграмме
- outputRenderer.java – класс, отрисовывающий элемент «выход» на диаграмме
- inputObject.java – класс для элемента «вход» на диаграмме
- inputRenderer.java – класс, отрисовывающий элемент «вход» на диаграмме
- controlObject.java – класс для элемента «управление» на диаграмме
- controlRenderer.java – класс, отрисовывающий элемент «управление»
- mechanismObject.java – класс для элемента «механизм» на диаграмме
- mechanismRenderer.java – класс, отрисовывающий элемент «управление»

- arrowRenderer.java – базовый класс для отрисовщиков стрелок
- activityBox.java – класс элемента «работа» на диаграмме
- activityBoxRenderer.java – класс, отрисовывающий элемент «работа»
- activityFrame.java – класс элемента рамки работы на диаграмме
- activityFrameRenderer.java – класс отрисовывающий рамку работы
- activePointList.java – класс коллекции активных точек
- badOrientationException.java – класс исключения неверного позиционирования
- moveEvent.java – класс события перемещения
- moveListener.java – интерфейс подписчика на событие перемещения
- resizeEvent.java – класс события изменения размера
- resizeEventListener.java – интерфейс подписчика на событие изменения
- reformEvent.java – класс события изменения пути стрелки
- reformEventListener.java – интерфейс подписчика на событие изменения пути
- projectManager.java – базовый класс, управляющий проектом
- project.java – класс проекта
- author.java – класс автора
- modelManager.java – базовый класс, управляющий моделью
- diagramTreeModel.java – класс дерева диаграммы
- diagram.java – класс диаграммы
- diagramManager.java – базовый класс, управляющий диаграммой
- modelElement.java – интерфейс элемента модели
- abstractModelObject.java – базовый класс элемента модели
- modelBox.java – базовый класс для работ модели
- activity.java – класс классической работы модели
- externalActivity.java – класс внешней работы
- modelArrow.java – базовый класс для стрелки
- input.java – класс элемента «вход»
- output.java – класс элемента «выход»
- control.java – класс элемента «управление»
- mechanism.java – класс элемента «механизм»
- configurationReader.java – класс, анализирующий конфигурационный файл
- customLoader.java – класс загружающий управляющие классы