Министерство науки и высшего образования Российской Федерации

Севастопольский государственный университет

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
К ЛАБОРАТОРНОЙ РАБОТЕ № 1
«ПРОВЕДЕНИЕ СИСТЕМНОГО АНАЛИЗА ПРЕДМЕТНОЙ ОБЛАСТИ, ОБЪЕКТА И МЕТОДА ИССЛЕДОВАНИЙ»

по дисциплине

«ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В НАУКЕ И ОБРАЗОВАНИИ»

для аспирантов дневной и заочной форм обучения

Севастополь, 2018

Методические указания к лабораторной работе № 1 «Проведение системного анализа предметной области, объекта и метода исследований» по дисциплине «Информационные технологии в науке и образовании» для аспирантов дневной и заочной форм обучения. Сост.: Мащенко Е.Н. - Севастополь, изд-во СевГУ, 2018. - 20 С.

СОДЕЖАНИЕ

1. ПОСТАНОВКА ЗАДАЧИ

В данной лабораторной работе необходимо последовательно применить принципы системного анализа к предметной области, объекту и методу исследований диссертационной работы по аналогии с примером, приведенным в данном указании. Для этого необходимо:

1) сформулировать цель исследования;
2) определить основное назначение (функции) исследуемого объекта (системы), что позволит определить его основные существенные свойства, показатели качества и критерии оценки;
3) создать системотехническое представление обьекта исследований в виде «черного ящика»: определить входные, выходные и управляющие параметры, привести перечень функций;
4) выполнить декомпозицию системы на подсистемы, модули, элементы;
5) определить входные, выходные и управляющие параметры, привести перечень функций для каждой из подсистем;
6) синтезировать (отобразить) внутреннюю структуру системы на функциональном и иерархическом уровнях;
7) при необходимости, описать вероятностный характер параметров объекта (системы).

Защита лабораторных работ проводится в виде докладов с использованием слайдов.

2. ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

2.1 Принципы системного анализа

Принципы системного анализа – это некоторые положения общего характера, являющиеся обобщением опыта работы человека со сложными системами. Различные авторы излагают принципы с определенными отличиями, поскольку общепринятых формулировок на настоящее время нет. Однако, так или иначе все формулировки описывают одни и те же понятия.

Наиболее часто, к системным причисляют следующие принципы: принцип конечной цели, принцип измерения, принцип единства, принцип связности, принцип модульности, принцип иерархии, принцип функциональности, принцип развития, принцип сочетания централизации и децентрализации, принцип учета неопределенности и случайностей. Пренебрежение этими принципами при проектировании любой нетривиальной технической системы, в том числе и программной, непременно приводит к потерям того или иного характера, от увеличения затрат в процессе проектирования до снижения качества и эффективности конечного продукта.

1) *Принцип конечной цели* – основополагающий принцип системного анализа. Конечная цель имеет абсолютный приоритет, в системе все должно быть подчинено достижению конечной цели. Принцип имеет несколько правил:

– для проведения системного анализа необходимо в первую очередь сформулировать цель исследования; расплывчатые, не полностью определенные цели влекут за собой неверные выводы (в этом смысле весьма красноречива расхожая фраза: «Четко сформулированная задача есть уже половина ее решения»);

– анализ следует вести на базе первоочередного уяснения основной цели (функции, основного назначения) исследуемой системы, что позволит определить ее основные существенные свойства, показатели качества и критерии оценки;

– при синтезе систем любая попытка изменения или совершенствования должна быть в первую очередь рассмотрена с позиции его полезности в достижении конечной цели;

– цель функционирования искусственной системы задается, как правило, системой, в которой исследуемая система является составной частью.

2) *Принцип единства*. Система должна рассматриваться и как целое, и как совокупность элементов. Расчленение системы (например, на подсистемы) необходимо производить, сохраняя целостное представление о системе. Принцип подразумевает выделение подсистем, композиция которых в совокупности со связями позволяет выполнять все функции проектируемой системы, определяет ее структуру и может быть рассмотрена как единая, целостная система.

3) *Принцип связности*. Любая часть системы должна рассматриваться со всеми своими связями с окружающими ее объектами, как внешними по отношению ко всей системе в целом, так и внутренними – другими элементами системы. Если некоторая подсистема имеет связи только с внешней средой, то есть смысл реализовать ее в виде отдельной системы. Подсистема, не связанная ни внешней средой, ни с другой подсистемой, является избыточной и должна быть удалена из системы.

4) *Принцип модульности.* В соответствии с этим принципом, выделение модулей системы, если таковые имеются, продуктивно и оправдано. Под модулями здесь понимаются относительно автономные и достаточно простые блоки, выполняющие ограниченный набор функций. Модуль в отличие от подсистем, которые имеют нерегулярную структуру и, как правило, несут определенную функцию, частично отражающую функцию системы, имеет регулярную структуру и характерные для него внутренние и внешние связи. Принцип указывает на возможность вместо части системы исследовать совокупность ее входных и выходных воздействий (абстрагирование от излишней детализации). Выделению модулей соответствует декопмозиция сложной задачи на множество более простых подзадач.

5) *Принцип иерархии*. Продуктивно выделять в системе иерархии, если они существуют. Иерархия [от гр. священная власть – порядок подчинения составных нижестоящих элементов и свойств вышестоящим по строго определенным ступеням (иерархическая лестница) и переход от низшего уровня к высшему] есть тип структурных отношений в сложных многоуровневых системах, характеризуемых упорядоченностью, организованностью взаимодействий между отдельными уровнями по вертикали. Иерархические отношения имеют место во многих системах, для которых характерна как структурная, так и функциональная дифференциация, т. е. способность к реализации определенного круга функций. Причем на более высоких уровнях осуществляются функции интеграции, согласования. Необходимость иерархического построения сложных систем обусловлена тем, что управление в них связано с переработкой и использованием больших массивов информации, причем на нижележащих уровнях используется более детальная и конкретная информация, охватывающая лишь отдельные аспекты функционирования системы, а на более высокие уровни поступает обобщенная информация, характеризующая условия функционирования всей системы, и принимаются решения относительно системы в целом. В реальных системах иерархическая структура никогда не бывает абсолютно жесткой в силу того, что иерархия сочетается с большей или меньшей автономией нижележащих уровней по отношению к вышележащим, и в управлении используются присущие каждому уровню возможности самоорганизации.[1]

6) *Принцип функциональности*. Принцип определяет первичность функции по отношению к структуре, так же, как цель первична для функции. Другими словами, цель определяет функции системы, а функции определяют ее структуру – совокупность элементов с их связями. Структура же, в свою очередь, определяет параметры системы. В случае придания системе новых функций полезно

пересматривать ее структуру, а не пытаться втиснуть новую функцию в старую схему. Поскольку выполняемые функции составляют процессы, то целесообразно рассматривать отдельно процессы, функции, структуры. В свою очередь, процессы сводятся к анализу потоков различных видов: материальный поток, поток энергии, поток информации, смена состояний. С этой точки зрения структура есть множество ограничений на потоки в пространстве и во времени.

7) *Принцип развития*. Это учет изменяемости системы, ее способности к развитию, адаптации, расширению, замене частей, накапливанию информации. В основу синтезируемой системы требуется закладывать возможность развития, наращивания, усовершенствования. Обычно расширение функций предусматривается за счет обеспечения возможности включения новых модулей, совместимых с уже имеющимися. С другой стороны, при анализе принцип развития ориентирует на необходимость учета предыстории развития системы и тенденций, имеющихся в настоящее время, для вскрытия закономерностей ее функционирования.

Одним из способов учета этого принципа разработчиками является рассмотрение системы относительно ее жизненного цикла. Условными фазами жизненного цикла ИС являются проектирование, изготовление, ввод в эксплуатацию, эксплуатация, наращивание возможностей (модернизация), вывод из эксплуатации (замена), уничтожение.

Отдельные авторы этот принцип называют принципом изменения (историчности) или открытости. Для того чтобы система функционировала, она должна изменяться, взаимодействовать со средой. На этом принципе основаны саморазвивающиеся и самоорганизующиеся системы.

8) *Принцип сочетания централизации и децентрализации.* Это сочетание в сложных системах централизованного и децентрализованного управления, которое, как правило, заключается в том, что степень централизации должна быть минимальной, обеспечивающей выполнение поставленной цели.

Недостаток децентрализованного управления – увеличение времени адаптации системы. Он существенно влияет на функционирование системы в быстро меняющихся средах. То, что в централизованных системах можно сделать за короткое время, в децентрализованной системе будет осуществляться весьма медленно.

Недостатком централизованного управления является сложность управления из-за огромного потока информации, подлежащей переработке в старшей системе управления. Поэтому в сложной системе обычно присутствуют два уровня управления. В медленно меняющейся обстановке децентрализованная часть системы успешно справляется с адаптацией поведения системы к среде и с достижением глобальной цели системы за счет оперативного управления, а при резких изменениях среды осуществляется централизованное управление по переводу системы в новое состояние.

9) *Принцип учета неопределенности и случайностей.* Принцип утверждает, что можно иметь дело с системой, в которой структура, функционирование или внешние воздействия не полностью определены.

Сложные открытые системы не подчиняются вероятностным законам. В таких системах можно оценивать «наихудшие» ситуации и рассмотрение проводить для них. Этот способ обычно называют методом гарантируемого результата. Он применим, когда неопределенность не описывается аппаратом теории вероятностей.

При наличии информации о вероятностных характеристиках случайностей (математическое ожидание, дисперсия и т.д.) можно определять вероятностные характеристики выходов в системе. В случае недостатка информации о поведении системы и взаимодействии ее с внешней средой следует предусматривать при функционировании системы реализацию способов пополнения информации о них

Принцип измерения. Система никогда не может быть объективно оценена средствами самой системы. О качестве функционирования какой-либо системы можно судить только применительно к системе более высокого порядка. Другими словами, для определения эффективности функционирования системы надо представить ее как часть более общей и проводить оценку внешних свойств исследуемой системы относительно целей и задач суперсистемы.

Перечисленные принципы обладают очень высокой степенью общности. Для непосредственного применения исследователь должен наполнить их конкретным содержанием применительно к предмету исследования. Такая интерпретация может привести к обоснованному выводу о незначимости какого-либо принципа. Однако знание и учет принципов позволяют лучше увидеть существенные стороны решаемой проблемы, учесть весь комплекс взаимосвязей, обеспечить системную интеграцию.

2.2 Интерпретация принципов системного анализа при проектировании программных систем

1) Принцип конечной цели

Как известно, проектирование прикладной программной системы начинается с анализа требований, которым она должна будет удовлетворять. Такой анализ проводится с целью понять назначение и условия эксплуатации системы настолько, чтобы суметь составить ее предварительный проект.

Моделью системы (или какого-либо другого объекта или явления) мы называем формальное описание системы, в котором выделены основные объекты, составляющие систему, и отношения между этими объектами. Построение моделей - широко распространенный способ изучения сложных объектов и явлений. В модели опущены многочисленные детали, усложняющие понимание. Моделирование широко распространено и в науке, и в технике.

Модели помогают:

- проверить работоспособность разрабатываемой системы на ранних этапах ее разработки;
- общаться с заказчиком системы, уточняя его требования к системе;
- вносить (в случае необходимости) изменения в проект системы (как в

начале ее проектирования, так и на других фазах ее жизненного цикла).

В основе метода системного анализа лежит принцип «черного ящика» (рис. 1). Его выходы определяются входами и внутренним строением (состоянием). В этом смысле говорят, что выход есть функция от входа и самого «черного ящика» Y=F(X,Z).

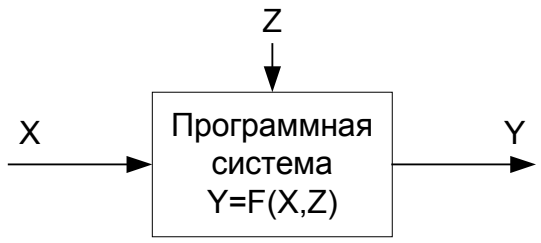

Рисунок 1 – Системотехническое представление программной системы в виде «черного ящика»

Входными данными системы является вектор X, управляющими параметрами - вектор Z (внутреннее состояние программной среды, настройки пользователя, содержимое базы данных и т.п.). Выходными данными системы служит вектор Y.

В соответствии с данным принципом должна быть четко сформулирована конечная цель – назначение проектируемой системы. На основании четко сформулированного назначения можно выделить функции системы, входные и выходные данные системы.

Таким образом, результатом применения принципа конечной цели являются список функций, которые должна выполнять система, а также спецификация входных и выходных данных в соответствии с ГОСТ 19.202-78.

2) Принцип единства

Согласно принципу единства, в программной системе выделяются подсистемы, каждая из которых выполняет полностью или частично некоторые функции проектируемой системы. Совокупность этих подсистем выполняет все функции системы. Отдельные подсистемы имеет смысл выделять для решения семантически слабо зависимых и достаточно крупных подзадач. В работах [2,3] не рекомендуется выделять слишком много подсистем, оптимальным можно считать выделение 3-7 подсистем. При выборе конкретного варианта декомпозиции проектируемой системы на подсистемы можно использовать один или несколько критериев (метрик) качества системы.

Результатом применения этого принципа является декомпозиция программной системы на подсистемы (модули, формы, библиотеки и т.п.).

3) Принцип связности

Определяются связи между подсистемами – смысловая нагрузка и направления передачи данных, которыми будут обмениваться выделенные подсистемы, а также подсистемы, которые будут обмениваться информацией непо-

средственно с внешней средой, т.е. принимать входные данные системы и формировать выходные.

4) Принцип модульности

Модульность – фундаментальный аспект всех успешно работающих крупных систем.

С увеличением объема программы становится невозможным удерживать в памяти все детали. Естественным способом борьбы со сложностью любой задачи является ее разбиение на более простые и обозримые части. Наиболее широко используемой технологией разработки программных систем на сегодняшний день является объектно-ориентированный подход к программированию (ООП). Объектно-ориентированный подход основан на систематическом использовании моделей для языково-независимой разработки программной системы.

Выделение *класса* является естественным развитием модульности. В классе структуры данных и функции их обработки объединяются. Класс используется только через интерфейс – детали реализации для пользователя класса несущественны. Идея классов отражает строение объектов реального мира – ведь каждый предмет или процесс обладает набором характеристик или отличительных черт, иными словами, свойствами и поведением. Программы часто предназначены для моделирования предметов, процессов и явлений реального мира, и классы являются удобным и адекватным инструментом для представления моделей в языке программирования.

Конкретные величины типа «класс» называются *экземплярами класса*, или *объектами*. Объекты взаимодействуют между собой, посылая и получая сообщения. Сообщение – это запрос на выполнение действия, содержащий набор необходимых параметров. Механизм сообщений реализуется с помощью вызова соответствующих функций. Таким образом, с помощью ООП легко реализуется так называемая «событийно-управляемая модель», когда данные активны и управляют вызовом того или иного фрагмента программного кода.

Класс содержит данные и функции для их обработки. Другим классам нежелательно иметь собственные средства обработки этих данных, они должны пользоваться для этого функциями первого класса. Для того чтобы использовать класс, нужно знать только его интерфейс, а не все детали его реализации. Чем более независимы модули, тем легче отлаживать программу. Скрытие деталей реализации называется *инкапсуляцией*. Инкапсуляция является ключевой идеей как структурного, так и объектно-ориентированного программирования. Интерфейсом модуля являются заголовки всех функций и описания доступных извне типов, переменных и констант.

В ООП идея инкапсуляции получила свое логическое завершение. Инкапсуляция повышает степень абстракции программы: данные класса и реализация его функций находятся ниже уровня абстракции, и для написания программы информация о них не требуется. Кроме того, инкапсуляция позволяет изменить

реализацию класса без модификации основной программы, если интерфейс остался прежним [2]. Реализации отдельных классов также рекомендуется помещать в отдельные модули.

Следующим шагом в повышении уровня абстракции программы является группировка классов в отдельные файлы (модули), компилируемые раздельно. Разбиение на модули уменьшает время перекомпиляции и облегчает процесс отладки, скрывая несущественные детали за интерфейсом модуля, позволяя отлаживать программу по частям (или разными программистами).

Результатом применения принципа модульности является декомпозиция подсистем и системы в целом на головную программу, модули, библиотеки.

5) Принцип иерархии

Построение иерархий, согласно принципам системного подхода к анализу задачи, может оказать существенную помощь в процессе формирования СПС.

Принцип иерархии в проекции на процесс разработки программных систем реализуется двумя базовыми свойствами ООП: наследованием и полиморфизмом.

Наследование – это возможность создания иерархии классов, когда потомки наследуют все свойства своих предков, могут их изменять и добавлять новые. Свойства при наследовании повторно не описываются, что сокращает объем программы. Выделение общих черт различных классов в один класс-предок является мощным механизмом абстракции.

Иерархия классов представляется в виде древовидной структуры, в которой более общие классы располагаются ближе к корню, а более специализированные – на ветвях и листьях.

Полиморфизм – возможность использовать в различных классах иерархии одно имя для обозначения сходных по смыслу действий и гибко выбирать требуемое действие во время выполнения программы.

Разработка иерархии классов является нетривиальной задачей. Грамотно спроектированные иерархии классов позволяют создавать высокоэффективные, легко читаемые и понимаемые программы. Плохо спроектированная иерархия приводит к созданию сложных и запутанных программ.

Важно до начала проектирования правильно определить, требуется ли вообще применять объектно-ориентированный подход. Если в иерархии классов нет необходимости, то, как правило, достаточно ограничиться модульной технологией [2].

Формальные признаки ситуации, когда выделение группы объектов в иерархию классов оправдано:

- наличие семантической связи между объектами;
- общие для объектов свойства, которые могут быть вынесены в базовый класс;
- общие для объектов методы – функции обработки данных, инкапсулированных в классе;

– различные в реализации, но семантически эквивалентные функции различных объектов могут быть включены в иерархию механизмов на базе полиморфизма, в частности, с помощью виртуальных методов, при этом в базовом классе может присутствовать только декларация метода, а реализация ложится на его потомков – такие базовые классы называются *абстрактными*.

Таким образом, с помощью механизмов ООП возможно объединение в иерархию объектов, различных по своей природе, но схожих по сути, т.е. на абстрактном уровне. Например, можно выделить в отдельную иерархию следующие объекты: файл, процесс, устройство и область памяти; определить для них базовый абстрактный класс, содержащий декларации методы чтения, записи и т.п., а в производных классах объектов соответствующим образом их реализовать. Это позволит формализовать обращение к этим объектам и оперировать ими одинаковым образом, абстрагируясь от их реальной природы.

Результатом применения принципа иерархии являются иерархии классов.

6) Принцип функциональности

На основании назначения подсистем и связей между ними для каждой из них определяются соответствующие им входные и выходные данные и функции.

Результатом применения принципов единства, связности и функциональности является структура (или множество возможных структур – тогда конечный вариант выбирается на этапе вариантного анализа) проектируемой системы и соответствующая ей структурная схема.

Основным требованием к синтезированной структуре является покрытие объединением множеств функций подсистем множества функций проектируемой системы, определенного применением принципа конечной цели.

7) Принцип развития

Принцип подразумевает учет возможности дальнейшего развития проектируемого программного продукта за счет:

- расширения предметной области;
- расширение функциональности;
- увеличения адекватности используемых моделей, которое влечет за собой возрастание интенсивности потоков обрабатываемой информации;
- переноса программного продукта на другие платформы;
- расширения и модернизации путем замены старых или добавления новых модулей, к примеру, на основе плагин-технологий;
- накопления информации о функционировании системы и ее анализа и т.п.

8) Принцип сочетания централизации и децентрализации

Примером применения данного принципа может служить разделение

пользовательского интерфейса управления системой на различные секции, разграничение прав доступа при администрировании системы для различных групп и отдельных пользователей, способ реакции системы на некоторое воздействие или событие (например, возможность напрямую обращаться к аппаратуре или жесткое регламентирование оного операционной системой через обращение к драйверу соответствующего устройства) и т.п.

9) Принцип учета неопределенности и случайностей

После выполнения данного принципа проектируемая система должна предусматривать определенную реакцию на неопределенные ситуации. Должны учитываться способы обработки некорректных входных данных, исключительных и аномальных ситуаций.

3 ПРИМЕР ПРИМЕНЕНИЯ ПРИНЦИПОВ СИСТЕМНОГО АНАЛИЗА ПРИ ПРОЕКТИРОВАНИИ ПРОГРАММНОЙ СИСТЕМЫ

3.1 Постановка задачи

Разработать программную систему трехзначного моделирования с задержками многоуровневых комбинационных схем на базе элементов И, ИЛИ, НЕ, И-НЕ, ИЛИ-НЕ. Обеспечить следующие возможности системы:

- создание и редактирование, сохранение в файл и последующая загрузка моделей логических элементов и схем;
- добавление и удаление моделей элементов в библиотеку элементов схемы;
- создание и редактирование совокупностей входных наборов для соответствующих им схем (входные сигналы могут принимать значения ‘0’, ‘1’ и ‘x’);
- результаты моделирования должны выдаваться и как конечные значения сигналов на выходах элементов схемы с соответствующими задержками, так и по требованию пользователя в виде временных диаграмм;
- сохранение и просмотр ранее сохраненных результатов.

Описание логического элемента включает:

- идентификатор (например, 2И-НЕ);
- тип (И, ИЛИ, НЕ, И-НЕ, ИЛИ-НЕ, тождественная единица, тождественный нуль);
- количество входов;
- задержки формирования сигналов ‘0’ и ‘1’ на выходе.

Описание схемы включает:

- идентификатор – имя схемы;
- количество входов и выходов;
- список цепей;

– список элементов схемы.

Элемент схемы может быть как логическим элементом, так и другой схемой, и характеризуется:

– идентификатором – функциональным именем;
– типом элемента – идентификатором логического элемента или схемы.

3.2 Системный анализ

Принцип конечной цели

Представим проектируемую программную систему в виде «черного ящика» (рис. 1), тогда входные данные – вектор X – будут включать в себя описание элементов, описание схемы и описание теста. Управляющие параметры системы – вектор Z – это режим выдачи результатов: сокращенная форма (только сигналы на выходах элементов схемы с указанием времени установления) или полная (с временными диаграммами для всех цепей схемы). Выходные данные – вектор Y – это результаты моделирования, представленные в соответствии с режимом их выдачи, а также библиотека моделей элементов схемы, модель схемы и соответствующая ей совокупность входных наборов, сохраненные в один файл, модели элементов, сохраненные в отдельные файлы.

Тогда для выполнения равенства Y=F(X,Z) проектируемая система должна выполнять следующие функции (в совокупности представляющие собой функцию F):

– автоматизированное построение и редактирование моделей логических элементов на основании их описания посредством пользовательского интерфейса;
– автоматизированное построение и редактирование модели многоуровневой комбинационной схемы на основании ее описания посредством пользовательского интерфейса;
– создание совокупности входных наборов для соответствующей схемы;
– сохранение и загрузка из файла определенного формата моделей элементов;
– сохранение и загрузка из файла определенного формата модели схемы, библиотеки ее элементов и совокупности соответствующих ей входных наборов;
– выполнение трехзначного моделирования работы схемы с задержками;
– построение временной диаграммы работы схемы на заданном входном наборе по результатам моделирования;
– просмотр, сохранение и загрузка из файла определенного формата результатов моделирования.

Принцип единства

На основании функций проектируемой системы, представленных выше, в ней можно выделить следующие подсистемы:

– подсистема создания моделей логических элементов;

- подсистема создания моделей схем;
- подсистема организации моделей логических элементов и подсхем схемы – библиотека элементов схемы;
- подсистема задания входных наборов и отображения результатов моделирования;
- подсистема трехзначного моделирования с задержками.

Принцип связности

Совокупность подсистем проектируемой программной системы и их связей – данными, которыми эти подсистемы обмениваются друг с другом и с внешней средой, – образует ее структуру. Структура проектируемой системы представлена на рис. 2.

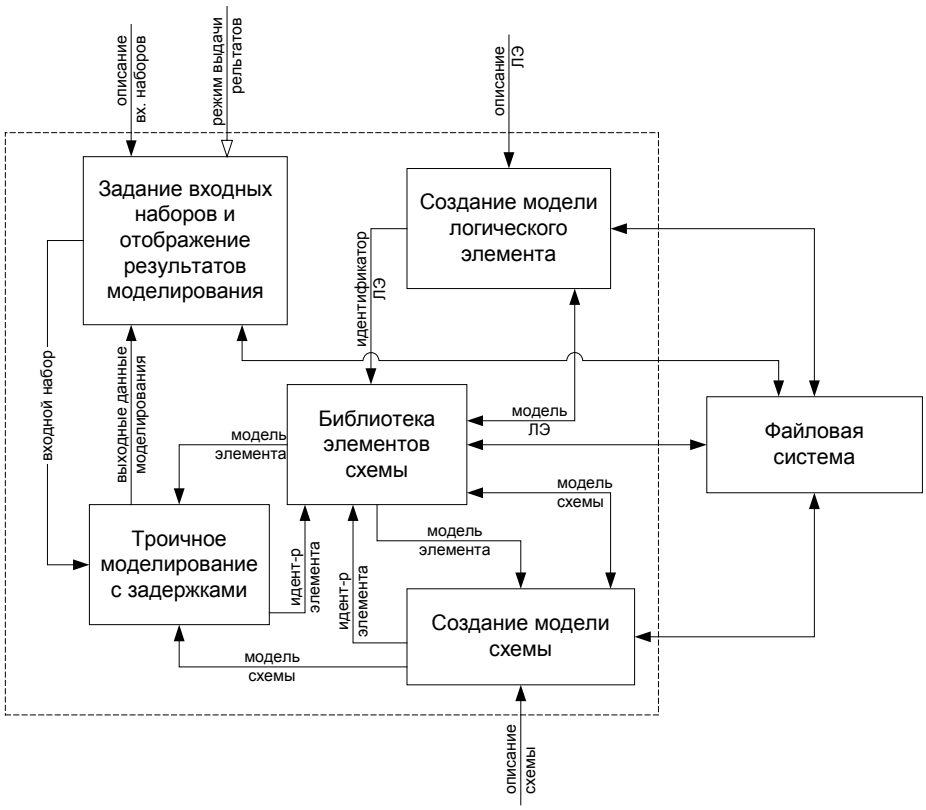

Рисунок 2 – Структура проектируемой системы

Принцип модульности

В проектируемой системе целесообразно выделить следующие модули:

- модуль логических элементов;
- модуль схем;
- модуль библиотеки элементов схемы;

– модуль элементов схемы (реализация классов элементов схемы, соответствующих логическим элементам, подсхемам и входам/выходам схемы как фиктивных логических элементов, для различного их отображения на схеме);

– модули интерфейса оператора (в том числе интерфейс задания входных наборов, интерфейс просмотра временных диаграмм, интерфейс создания модели логического элемента, интерфейс создания модели схемы и т.д.).

Отсутствие отдельного модуля моделирования станет понятным чуть ниже.

Принцип иерархии

Согласно этому принципу в сочетании с методологией объектно-ориентированного программирования, в проектируемой системе можно выделить две иерархии объектов (рис.3 и 4).

Вертикальные связи в совокупности с вершинами (классами) образуют дерево наследования – иерархию программных классов. Методы, выделенные жирным курсивом, являются абстрактными. Все переопределяемые методы являются виртуальными.

На рисунке 3 представлена иерархия классов, соответствующая моделям логических элементов и схем. Пунктирными линиями обозначены семантические связи между элементами и библиотекой элементов схемы.

Класс TBaseElem является базовым абстрактным классом, и определяет основное предназначение всех производных от него классов – выполнять моделирование работы соответствующего ему объекта. Поле ID представляет идентификатор объекта, nIn и nOut – количество входов и выходов соответственно. Метод Simulate выполняет моделирование работы соответствующего объекта, на заданном наборе входных сигналов и возвращает значения выходных сигналов, а также время, за которое эти сигналы установились в свое конечное значение. В данном классе он является абстрактным. Методы Load и Save здесь и далее осуществляют загрузку и сохранение объекта в поток.

Класс TLogicElem соответствует модели абстрактного логического элемента. Поля t01 и t10 определяют задержки перехода сигнала на единственном выходе (nOut=1) элемента из состояния ‘0’ в ‘1’ и из ‘1’ в ‘0’ соответственно (время перехода из состояния ‘x’ в $z \in \{‘0’,’1’\}$ определяется как время перехода из $\bar{z}$ в z).

Классы TAnd, TOr, TNAnd и TNOr реализуют модели логических элементов И, ИЛИ, И-НЕ и ИЛИ-НЕ соответственно (элемент НЕ может быть реализован как элемент И-НЕ или ИЛИ-НЕ с одним входом). Классы соответствующим образом переопределяют метод Simulate.

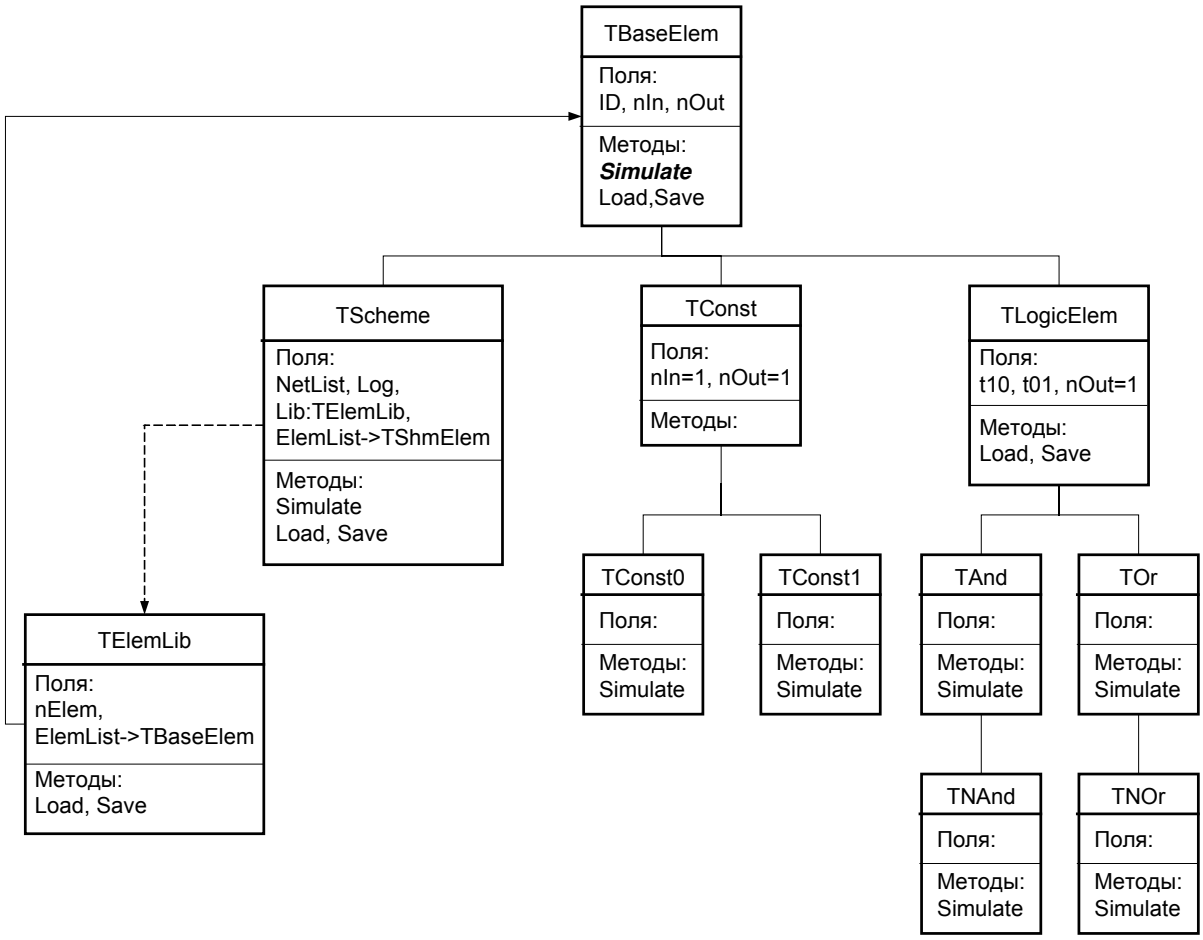

Рисунок 3 – Иерархия классов моделей элементов

Класс TConst соотвествует модели абстрактной константной неисправности (тождественному значению сигнала), а TConst0 и TConst1 моделям элементов с сигналами ≡0 и ≡1 на выходе.

Класс TScheme реализует модуль схемы. Поле NetList определяет список цепей, поле ElemList список элементов (ссылок на объекты из библиотеки элементов схемы), поле Lib – библиотека элементов схемы, поле Log – служебная информация моделирования, используется при построении временных диаграмм (может представлять из себя, например, массив длиной, равной количеству цепей в схеме, элементы которого являются списками записей о появлении в соответствующей цепи определенного значения сигнала в определенный момент времени). Реализация метода Simulate определяется выбором алгоритма моделирования (например, синхронное или асинхронное, событийное или несобытийное).

Класс TElemLib соответствует библиотеке элементов схемы. Поле nElem определяет количество элементов в библиотеке, поле ElemList – список элементов (производных от TBaseElem).

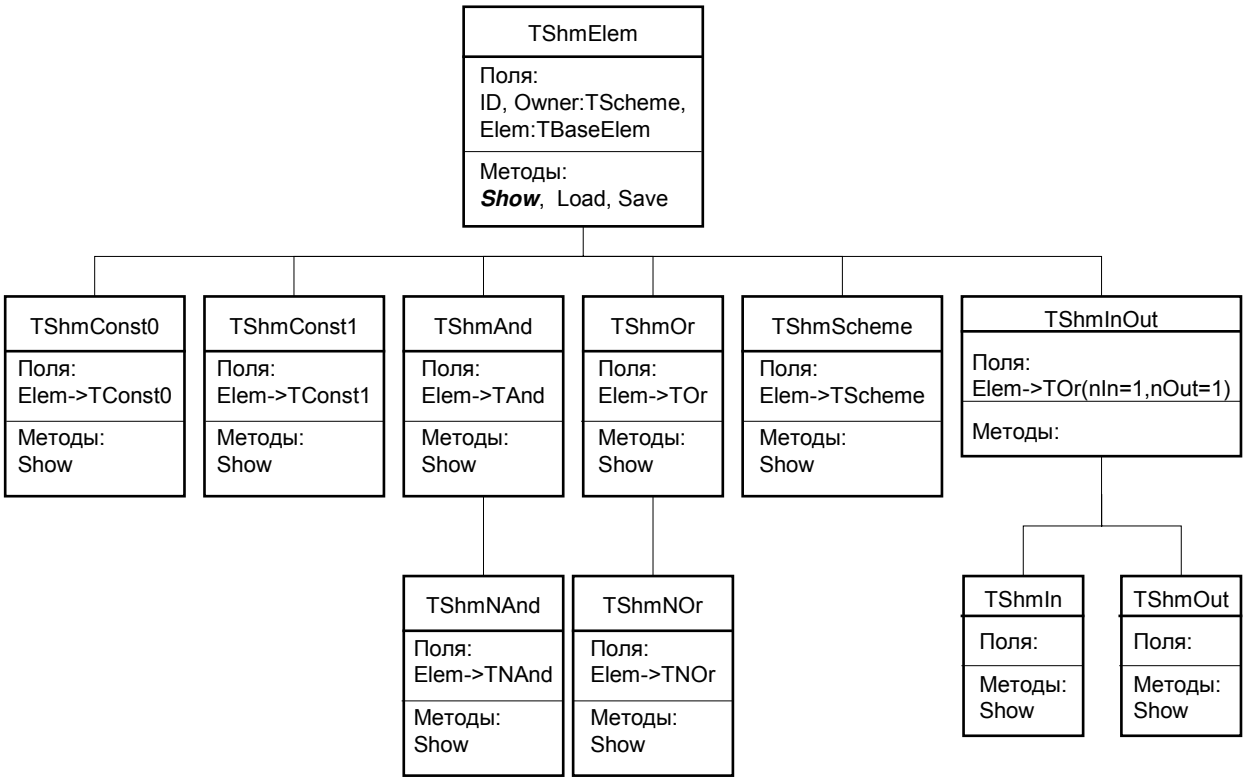

Рисунок 4 – Иерархия классов элементов схемы

На рисунке 4 представлена иерархия классов элементов схемы.

Класс TShmElem является базовым классом и соответствует абстрактному элементу схемы. Основное назначение иерархии – обеспечить механизм хранения информации о типе элемента и отображение элементов на схеме в интерфейсном окне оператора в соответствии с типом элемента. Поле ID определяет функциональное обозначение элемента на схеме, поле Owner – ссылка на схему, к которой относится элемент, поле Elem – ссылка на элемент библиотеки элементов схемы, который и определяет тип. Метод Show предназначен для отображения элемента на схеме, он объявлен абстрактным, но это не обязательно: некоторые действия по отображению элемента вполне могут быть выполнены и на уровне базового класса.

Классы TShmConst0, TShmConst1, TShmAnd, TShmOr, TShmNAnd, TShmNOr, TShmScheme реализуют следующие элементы схемы (в порядке перечисления): ≡0, ≡1, И, ИЛИ, И-НЕ, ИЛИ-НЕ, подсхему.

Класс TShmInOut соответствует фиктивному элементу схемы – внешнему выводу схемы (вывод разъема). Его поле Elem ссылается на повторитель (одновходовой элемент ИЛИ). Этот выбор продиктован не общими соображениями, а, скорее, связан с особенностями выбранного автором алгоритма моделирования, структур данных и т.п. Возможны и другие решения, например, не создавать отдельных элементов для каждого вывода разъема, а один общий для всего разъема, с соответствующим количеством выводов, такое решение может предполагать создание отдельного класса в иерархии моделей элементов. Классы TShmIn и TShmOut соответствуют входному и выходному выводу схемы.

Принцип функциональности

Функции системы в целом рассмотрены в связи с принципом конечной цели. Рассмотрим функции, входные и выходные данные выделенных подсистем.

Основной функцией подсистемы создания моделей логических элементов является, очевидно, создание моделей логических элементов (ЛЭ) на основе данных, вводимых оператором, – описания ЛЭ. Описание ЛЭ есть входные данные для подсистемы, на выходе системы – модель ЛЭ. Поскольку подсистема должна позволять редактирование ранее созданных моделей, то подсистема может также может принимать от библиотеки элементов схемы ранее созданные модели ЛЭ.

Подсистема создания моделей схем должна позволять оператору создавать и редактировать модели схем на основании описания схемы и действий оператора по ее разметке. Выходные данные подсистемы – модель схемы. Элементы схемы также могут быть схемами, то для таких элементов возможно редактирование модели подсхемы, переданной из библиотеки элементов схемы.

Подсистема задания входных наборов и выдачи результатов тестирования дает возможность оператору задать необходимое число входных наборов, инициировать моделирование заданной схемы на этих наборах и проанализировать его результаты на основе конечных значений сигналов на выходных вентилях элементов схемы и времени их установления, а также временных диаграмм работы схемы на заданном входном наборе. Режим отображения результатов моделирования является управляющим параметром подсистемы. Входные данные системы – описание входных наборов, выходные данные подсистемы моделирования, выходные данные подсистемы – входные наборы, передаваемые подсистеме моделирования, конечные значения сигналов на выходных вентилях элементов схемы, время их установления, а также временные диаграммы.

Библиотека элементов схемы определяет элементный базис построения схемы и организует доступ к моделям элементов. На вход библиотеки поступает идентификатор элемента (или подсхемы), а на ее выходе – модель элемента (или подсхемы).

Подсистема моделирования выполняет трехзначное моделирование с задержками. Входными данными для нее являются модель схемы, модели элементов схемы и входной набор, на котором будет производиться моделирование. Выходные данные – это информация о состоянии цепей схемы на протяжении всего интервала моделирования.

Все подсистемы, за исключением подсистемы моделирования, также обмениваются данными с файловой системой при загрузке и сохранении соответствующих данных.

Принцип развития

Проектируемая система может быть расширена следующим образом:

– расширение элементной базы (например, добавление моделей более сложных комбинационных устройств включением нового класса в иерархию моделей элементов);
– расширение предметной области, т.е. моделирование схем с обратными связями (за счет усложнения алгоритма моделирования);
– введение возможности работы с моделями и результатами моделирования не только на уровне файлов, но и с использованием централизованной базы данных.

Принцип сочетания централизации и децентрализации

В множестве выделенных подсистем можно выделить несколько подмножеств (возможно пересекающихся), которые будут обладать достаточно высокой степенью автономности от других подмножеств. Например, можно выполнить декомпозицию таким образом:
– {подсистема создания моделей логических элементов};
– {подсистема создания моделей схем, библиотека элементов схемы};
– {подсистема создания задания входных наборов и выдачи результатов моделирования, подсистема моделирования, подсистема представления модели схемы (как часть подсистемы создания моделей схем), библиотека элементов схемы}.

Такое разбиение позволит реализовать полученные подмножества в виде отдельных исполняемых модулей и физически разделить процессы создания моделей ЛЭ, схем и моделирования в проектируемой системе. В этом случае обмен данными между подсистемами разных подмножеств будет производиться через файловую систему.

С другой стороны, все подсистемы можно реализовать в одном исполняемом модуле, регламентируя порядок обращения к каждой из подсистем посредством интерфейса оператора.

Принцип учета неопределенности и случайностей

В проектируемой системе следует предусмотреть возможность реакции на некорректные с точки зрения системы действия оператора, например:
– попытка повторного использования функционального обозначения элемента;
– несовместимость имеющейся и подгружаемой из файла модели ЛЭ или подсхемы по каким-либо параметрам (к примеру, числу входов);
– попытка создания обратных связей и т.д.

3.2.1 БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Спицнадель В. Н. Основы системного анализа. – СПб.: «Издательский дом «Бизнес-пресса», 2000.
2. Сурмин Ю.П. Теория систем и системный анализ: Учеб. пособие. К.: МАУП, 2003. 368 с.