# Аннотация

Данный документ содержит всестороннее исследование посылок и описание основных этапов реализации универсального программного комплекса для моделирования технических устройств смешанной природы «Программа Анализа», версии 10 (далее ПА10).

ПА10 предназначена для анализа динамики различных (механических, гидравлических, пневматических, тепловых и смешанных) технических систем, функционирование которых описывается системой ДАУ. Областью применения данного продукта является проектирование и анализ промышленных изделий и систем, объединяющих в себе функциональные элементы различного типа, начиная от систем защиты и контроля, электрических схем систем управления, до бытовой техники.

Данная версия комплекса изначально разрабатывается как ориентированная на визуальное проектирование, и должна отличаться дружественным интерфейсом, широкими функциональными возможностями, масштабируемостью и гибким механизмом специализации под конкретные задачи проектирования.

Одним из будущих применений разрабатываемой системы является использование её для проведения лабораторных работ по курсу «Основы Автоматизированного Проектирования» с целью обучения студентов современным методикам проектирования технических устройств и анализа их динамических характеристик.

# Содержание

[Аннотация 1](#_Toc105756068)

[Содержание 2](#_Toc105756069)

[Введение 4](#_Toc105756070)

[Исследовательская часть 5](#_Toc105756071)

[Архитектура и дизайн 5](#_Toc105756072)

[Введение 5](#_Toc105756073)

[Архитектура и строительство 7](#_Toc105756074)

[Биологический императив 8](#_Toc105756075)

[Историческая справка 12](#_Toc105756076)

[Психологические аспекты разработки ПО 16](#_Toc105756077)

[Поток исполнения 16](#_Toc105756078)

[Элементы правильного дизайна 17](#_Toc105756079)

[Специализированные языки 19](#_Toc105756080)

[Введение 19](#_Toc105756081)

[Цена вопроса 20](#_Toc105756082)

[Языки существующих систем 20](#_Toc105756083)

[Технологическая часть 26](#_Toc105756084)

[Архитектура современного ПО 26](#_Toc105756085)

[Компонентная технология 27](#_Toc105756086)

[Управляемый код 29](#_Toc105756087)

[Дизайн современного ПО 29](#_Toc105756088)

[Практическая часть 31](#_Toc105756089)

[Архитектура комплекса ПА10 31](#_Toc105756090)

[Введение 31](#_Toc105756091)

[Узлы декомпозиции 31](#_Toc105756092)

[Дизайн комплекса ПА10 36](#_Toc105756093)

[Дизайн-тайм компоненты ПА10 36](#_Toc105756094)

[Страница редактирования 37](#_Toc105756095)

[Визуальный редактор функциональных эквивалентных схем (ФЭС) 37](#_Toc105756096)

[Понятие страницы редактирования 37](#_Toc105756097)

[Редактирование схем в ПА10 38](#_Toc105756098)

[Работа с компонентами 39](#_Toc105756099)

[Работа с решениями 40](#_Toc105756100)

[Редакторы свойств 41](#_Toc105756101)

[Трассировка проводников 42](#_Toc105756102)

[Сравнение возможностей систем серии ПА 43](#_Toc105756103)

[Огранизационно-экономическая часть 44](#_Toc105756104)

[Расчет трудоемкости разработки первой версии комплекса ПА10 44](#_Toc105756105)

[Расчет стоимости основных производственных фондов, используемых для разработки первой версии комплекса ПА10 45](#_Toc105756106)

[Расчет затрат на разработку первой версии комплекса ПА10 46](#_Toc105756107)

[Материалы, покупные изделия и полуфабрикаты 46](#_Toc105756108)

[Специальное оборудование 46](#_Toc105756109)

[Заработная плата 46](#_Toc105756110)

[Амортизационные отчисления 47](#_Toc105756111)

[Контрагентные работы 48](#_Toc105756112)

[Производственные командировки 48](#_Toc105756113)

[Косвенные расходы 48](#_Toc105756114)

[Расчет налогов 48](#_Toc105756115)

[Себестоимость разработки первой версии комплекса ПА10 49](#_Toc105756116)

[Формирование расчетной (остаточной) прибыли предприятия и определение эффективности произведенных затрат на разработку ПАК для проведения лабораторных работ на ЭВМ 49](#_Toc105756117)

[Выводы. 51](#_Toc105756118)

[Промышленная экология и безопасность 52](#_Toc105756119)

[Введение. 52](#_Toc105756120)

[Анализ условий труда. 52](#_Toc105756121)

[Требования к микроклимату. 52](#_Toc105756122)

[Освещение рабочего места. 52](#_Toc105756123)

[Расчет вентиляции 55](#_Toc105756124)

[Эргономические требования к дисплею 59](#_Toc105756125)

[Электробезопасность компьютера. 59](#_Toc105756126)

[Столы и стулья, используемые для оборудования рабочего места. 60](#_Toc105756127)

[Требования к рабочему столу 60](#_Toc105756128)

[Выводы по разделу. 60](#_Toc105756129)

[Приложения 62](#_Toc105756130)

[Приложение 1. Google Citation Index 62](#_Toc105756131)

[Приложение 2. Пример кода сборки и решения эквивалентной схемы на C#. 65](#_Toc105756132)

[Приложение 3. Пример сборки схемы на XML 66](#_Toc105756133)

# Введение

Комплекс ПА10 представляет собой CAE систему моделирования динамических процессов в технических объектах на сосредоточенном уровне, предназначенную для анализа процессов в гетерогенных физических системах во временной и частотной области. Имеются возможности для комплексного моделирования технических объектов, состоящих из различных физических подсистем - электрических, механических, гидравлических, тепловых и так далее. В одну группу с ПА10 входят такие системы как MSC.ADAMS, и предшествующие разработки серии ПА. Моделирование на сосредоточенном уровне находит все большее применение при проектировании изделий промышленности, т.к. позволяет получить точные результаты за приемлемое время и без использования больших вычислительных ресурсов.

В предыдущих версиях комплекса разработчикам не удалось в полной мере реализовать вышеперечисленные требования к подобным системам. Версия 7 характеризовалась высокой функциональностью, возможностью расширения набора моделируемых объектов и гибкостью, за счет использования C-подобного языка моделирования. Однако, она была написана под DOS и к настоящему моменту уже морально устарела, т.к. ей недостает графического интерфейса пользователя и поддержки множества современных технологий, таких как клиент-сервер и COM.

Версия 9 была ориентирована на программную переносимость и дружественность рабочей среды, но, в силу многих недостатков, не обладала достаточной гибкостью и функциональностью, а также, что самое главное, эффективностью проведения расчетов.

Таким образом, основной задачей при разработке версии 10 можно считать устранение изученных недостатков предыдущих систем, а также внедрение новых, более эффективных и точных средств математического моделирования.

В комплексе ПА10 пользователь, являющийся специалистом в своей предметной области, но не знакомый с технологиями программирования, имеет возможность создания полноценных моделей в визуальной среде. Код классов моделей генерируется средой разработки, компилируется и включается в пакет.

В качестве языка разработки ПА10 выбран основной язык Net - C#. Выбор программной технологии должен учитывать скорость создания прототипов и последующей разработки, а также возможности интеграции. На текущий момент одной из самых доступных и богатых по возможностям технологий является технология .Net от компании Microsoft. Обоснования принятых проектных решений и основные этапы на пути их реализации и составляют предмет данного дипломного проекта.

# Исследовательская часть

## Архитектура и дизайн

### Введение

В любой деятельности человек, прежде всего, стремиться к тому, чтобы в основе её лежали правильные соображения. Результатом «правильных соображений» может оказаться что угодно. У каждого свои взгляды на то, что считать правильным, и как правильно вести дела. Это и становится причиной большинства конфликтов. Программные системы не являются исключением.

Когда говорят о правильных тенденциях в программном обеспечении, используют понятия *архитектуры* и *дизайна*.

Современный взгляд на архитектуру и дизайн программных систем состоит в том, что это в чем-то пересекающиеся понятия. Принято считать, что не существует единых и общепризнанных определений дизайна и архитектуры. В разных источниках встречается различные определения. От совсем простых:

Architecture is the organizational structure of a system.

IEEE Std. 610.12-1990

Архитектура – это организационная структура системы.

Стандарт IEEE 610.12-1990

Design – (1) The process of defining the architecture, components, interfaces, and other characteristics of a system or component. (2) The result of the process in (1).

IEEE Std. 610.12-1990

Дизайн – (1) Процесс задания архитектуры, компонентов, интерфейсов и других характеристик системы или компонента. (2) Результат этого процесса в (1).

Стандарт IEEE 610.12-1990

И расширенных:

The software architecture of a program or computing system is the structure or structures of the system, which comprise software components the externally-visible properties of those components and the relationships among them.

Len Bass, Paul Clements, Rick Kazman: *Software Architecture in Practice*, Addison-Wesley, 1998

Архитектура программной или вычислительной системы – это структура системы (или набор структур, её составляющих), которая включает в себя программные компоненты, видимые извне свойства этих компонентов и отношения между ними.

Лен Басс, Пол Клеменс, Рик Кацман, *«Программная архитектура на практике»*, Addison-Wesley, 1998

Design is the activity of making explicit proposals for a change from an existing system to some future system which more closely approximate   
mankind's concept of the ideal.

Tom Maver, Professor of Stathclyde University, Glasgow

Дизайн – это деятельность по выдвижению явных предложений о переходе от существующей системы к некоторой будущей системе, которая бы лучше соответствовала человеческому пониманию идеала.

Том Мавер, профессор университета Статклайд в Глазго

До совсем уж сложных:

Software architecture is a set of concepts and design decisions about the structure and texture of software that must be made prior to concurrent engineering to enable effective satisfaction of architecturally significant explicit functional and quality requirements and implicit requirements of the product family, the problem, and the solution domains.

Jazayeri, Ran, van der Linden: *Software Architecture for Product Families*: Principles and Practice, Addison Wesley Longman, 2000

Программная архитектура – это набор понятий и проектных решений в отношении структуры и строения программного обеспечения, которые должны быть приняты до начала параллельного проектирования, чтобы обеспечить эффективное удовлетворение внешних функционально значимых и качественных архитектурных требований, а также безусловных требований к данному семейству продуктов, проблемных областей или решений.

Джэзаери, Ран, Ван дер Линден, «Программная архитектура для семейств продуктов: принципы и практика», Addison Wesley Longman, 2000

Design is the process of applying various techniques and principles for the purpose of defining a device, a process, or a system in sufficient detail to permit its physical realization.

Roger S. Pressman: Software Engineering: A Practioners Approach, McGraw-Hill, 1992

Дизайн – это процесс применения различных техник и принципов в целях задания устройства, процесса или системы в детализации, достаточной для их физической реализации.

Роджер С. Прессман, «Разработка программного обеспечения: подход практиков», McGraw-Hill, 1992

Пересечение понятий архитектуры и дизайна проявляется в ответе на вопрос о том, что же конкретно является результатом применения дизайна. Учитывая, что дизайн – это неотъемлемое свойство системы, порожденное применением практик определенного дизайна, довольно сложно сформулировать, как этот результат соотносится с архитектурой. К примеру, использование классов – это дизайн или архитектура? Проект начинается с архитектуры, но и на стадии завершения она всё еще с нами, только теперь может отличаться от первоначальной.

И хотя для большого количества понятий можно достаточно точно разделить, что в них относится к дизайну, а что влияет на архитектуру, остается множество вещей, доказательство принадлежности которых к тому или другому – сплошная софистика.

В то же время, разделение проектных решений на архитектурные и дизайнерские, вопрос не праздный. В отношении дизайна, как результата, бытует мнение, что дизайн – это архитектура в малом. И если архитектурой призваны заниматься эрудированные умники с чистыми волосами, то дизайн – удел странноватых программистов. Так принято. Часто случается, что архитектор системы самоустраняется сразу же после утверждения основных диаграмм и переходит на другой проект, и на другой проект, и на другой проект, оставляя непродуманные решения на совесть программистов. Часто случается, что программисты убивают на корню любую, даже самую продуманную, архитектуру, внося ужасные ошибки и делая неумные дизайнерские ходы.

Также часто можно услышать, что где-то пожертвовали дизайном ради эффективности (под которой понимается скорость исполнения алгоритмов), или ради скорости разработки. Жертвуют ли архитектурой?

Архитектурой жертвовать нельзя. Как следует из определений, архитектура задается еще до начала параллельной разработки. Соответственно, нарушение архитектурных соглашений легко может привести к тому, что проект рассыплется, как Шалтай-болтай. Из другого определения видно, что архитектура – это набор свойств и соглашений, видимых извне. Нарушения в данной области приведут к тому, что продукт принципиально не сможет быть интегрирован с окружением и другими системами так, как это задумывалось. С такой точки зрения, разделение на архитектуру и дизайн также означает приоритетность определенных установок для команды проекта. Вероятно, архитектуру можно определить и так:

Проектные решения, которыми жертвуют только при пересмотре взглядов на проект в целом.

И в отличие от ошибок в программировании, возможно порожденных неправильным дизайном, архитектурные ошибки стоят разработчикам гораздо дороже первых.

Не имея архитектуры, кажется нельзя даже подступиться к проекту. По крайней мере, очень сложно вести параллельную разработку. Видимо, архитектура – первое, что приходит в голову.

В то же самое время, дизайн имеет отношение к выбранному языку, инструментам и программному коду. Дизайн определяет культуру программирования и квалификацию разработчиков.

Поэтому архитектор, приступая к проекту, должен сам для себя ограничить меру ответственности, чтобы не заступать на территорию дизайна и не тормозить тем самым ход разработки.

### Архитектура и строительство

Вспоминаю, как в первый день руководитель отдела кадров знакомил меня с сотрудниками, представляя: «Наш новый главный архитектор». К концу того же дня один из моих новых коллег обратился ко мне, плохо понимая, в чем состоят мои обязанности. «В общем и целом, — заметил он, разглядывая стены и потолок, — здание выглядит неплохо. И что Вы намерены с ним делать?»

Стив Виноски, *«Все начинается с архитектуры»*, Открытые системы, №1, 2004

В общем и целом, архитектура программного обеспечения (ПО), также как и архитектура здания, призвана сделать продукт эстетичным, прочным и долговечным. Архитектура должна не позволить развалиться проекту, открытому всем ветрам изменений, и найти уязвимые места, даже если «здание выглядит неплохо». Архитектура не должна быть слишком сложной для имеющихся средств реализации. А также понятной для тех, кто будет строить.

Сравнение процесса создания программ с процессом возведения сооружений очень глубоко проникло в умы и отразилось в терминологии. Употребляются такие слова как architecture[[1]](#footnote-1), build[[2]](#footnote-2), framework[[3]](#footnote-3) и т.д. Очевидно, что подобное сравнение импонирует разработчикам. Вот некоторые особенности современного строительства, которые очень хотелось бы видеть в разработке ПО:

* Всё строится из *готовых* стандартных блоков;
* Много *формальных* методов для проектирования и собственно строительства;
* Наличие большого числа метрик для *объективной* оценки качества;
* *Предсказуемость* процесса во времени и затратах;

Ничто из этого на момент 2005 года не имеет ничего общего с разработкой ПО. Сравнение со строительством также переносит на программное обеспечение некоторые сомнительные следствия:

* **Каждое здание имеет фундамент** – значит, каждая программная система должна иметь свой фундамент, своё «ядро». Разработка такого ядра может легко занять 200% времени, выделенного на разработку всего продукта;
* **Строительство зданий дорогое занятие** – значит, разработка ПО должна стоить очень дорого;
* **Строители не слишком образованы** (в отличие от архитекторов) – значит, и для ПО можно нанять много очень недорогих и неквалифицированных программистов, которые сделают качественный и крепкий продукт под руководством одного хорошего менеджера.

Неверные представления подтверждают, что строительство – не лучшая аналогия для процесса создания программного обеспечения. Возможно, проектирование планов сооружений является более близкой аналогией. На самом же деле работа инженера-строителя больше похожа на работу архитектора от программного обеспечения, т.к. планы создаются *до начала строительства*, и для строящегося здания размеры фундамента больше не пересматриваются. В индустрии программного обеспечения сплошь и рядом можно встретить требования по расширению/углублению фундамента уже на стадии завершения проекта. Таким образом, строительная аналогия неадекватна с любой точки зрения.

### Биологический императив

У программных систем существует одно очень существенное отличие от всех других, реализующихся в материи, игнорировать которое невозможно. Любая модель программной системы сама по себе является программной системой. Моделью физической системы (наравне с уменьшенной физической моделью), может служить система уравнений. Для программной системы также могут существовать диаграммы, но это совсем не модель, а скорее документация[[4]](#footnote-4).

Видимо, можно говорить, что *архитектура* является некой моделью программной системы. Но даже здесь правильнее говорить не о модели, а о некоторой проекции настоящей модели системы, работающей в голове архитектора. Та, настоящая модель, очевидно, - программа. Хотя возможно, что и эта проекция в виде программной архитектуры является особым видом декларативной программы.

Данное отличие (рекурсивность программ и их моделей) порождает свои проблемы и свои преимущества.

Вот основные проблемы:

1. Создание адекватной модели сопоставимо по затратам с созданием самой системы, поэтому искушение перенести модель в реальный проект часто побеждает опыт и здравый смысл;
2. Моделирование способно утонуть в деталях реализации и перестать быть моделированием;

А ниже перечислены некоторые важные преимущества:

1. Можно совсем не моделировать систему (за исключением прототипов пользовательского интерфейса);
2. Инструменты изучения модели те же, что и для проверки работы готовой системы – это удешевляет разработку;

Преимущество за номером один является самым важным. Оно не означает, что можно забыть о проектировании и сразу кидаться головой в прорубь. Оно означает, что программную систему можно как бы *выращивать*, создавая из малого большое так, что на каждом этапе мы имеем работоспособную конструкцию. Военные спят и видят такие ракеты и танки, которые можно было бы просто выращивать на деревьях и бросать в бой в любой момент, еще недозревшими, если нужна быстрая мобилизация. Фантастика!

Игнорирование приведенного полезного свойства ни к чему хорошему не приводит.

К слову сказать, в программном мире еще достаточно «Титаников» от программного обеспечения, которые нельзя пустить на воду до тех пор, пока не будет поставлена последняя, 10-ти миллионная заклепка[[5]](#footnote-5). Первый же серьезно настроенный айсберг сводит на нет все усилия пары тысяч программистов[[6]](#footnote-6).

Такая нерасторопность не поощряется на рынке современного программного обеспечения. Поэтому исследования новых методов инкрементальной разработки всегда вызывают большой интерес. К тому же они приносят неплохую прибыль[[7]](#footnote-7).

Программирование действительно является очень сложной областью, в которой высок уровень неудовлетворительных результатов. Вот примерная статистика успешности программный проектов на 2000 год[[8]](#footnote-8):

* Успешны – 28%;
* Провалились – 23%;
* Превысили время и/или бюджет – 49%.

Поэтому архитектурные споры не утихают ни по одному пункту вопроса. Вот некоторые примеры:

* Объектно-ориентированное программирование (ООП) против процедурно-ориентированного программирования (ПОП);
* Объектные базы данных (OODB) против реляционных баз данных (RDB);
* Сервера приложений (Application Servers) против архитектуры, основанной на сервисах (Service-Oriented Architecture - SOA).

Аргументы, приводимые сторонами, часто сводятся к следующему: я программирую, используя замечательный X, а курица программирует, используя этот ужасный Y. Представляете, какой ужасный продукт получится у курицы? Т.е. решающими в аргументах оказываются не свойства парадигмы, а профессионализм в применениях.

В действительности, более эффективными являются та парадигма и та архитектура, которые выживут в итоге. Т.е. заранее знать нельзя, но это также даёт и ключ к анализу: оценивайте источники, питающие тот или иной подход. Как минимум, хорошая архитектура должна:

1. Быть совместимой с предыдущей популярной архитектурой;
2. Нравиться людям;
3. Иметь хорошо различимые пути эволюции.

В данном тексте проводится линия сравнения производства ПО с живыми системами. Аргументация применимости такого сравнения не намного лучше, чем у строительства. Довольно резонно, что настоящий документ о новом взгляде на известные вещи будет так или иначе «развенчивать устаревшие мифы», и пропагандировать мифы новые. В сущности, у программ нет почти ничего общего с живыми системами. Приводимый ниже аргумент больше схож со слабым антропным принципом[[9]](#footnote-9) для космогонических теорий. Аргумент такой:

Если живые системы на данном этапе не идут ни в какое сравнение с другими типами систем, значит это и есть лучшее, что можно получить в данных условиях.

Это не означает, что живые системы лучшие в принципе. Это означает только, что в данных условиях, при данном количестве доступной энергии и существующей агрессии внешней среды, живые системы показывают лучшую организацию, стабильность и развитие. При прочих равных, лучше проводить аналогии с тем, что доказывает свою большую эффективность. И заимствовать оттуда же. Ведь живые системы лучше строений и машин по многим важным параметрам.

Чтобы лучше обосновать такую точку зрения, ниже приводится список аналогий между биологической жизнью и программным обеспечением, следствия из которых были бы полезны для последнего.

#### Код и данные

Живые системы разделяют код ДНК и живое тело, им порождаемое. Программные системы живут в памяти. При старте они «разворачиваются» в ней, порождая данные. И для живых систем, и для программ, код и данные занимают часть общего пространства. Вероятно, первым понятие программ, хранимых в той же памяти, что и данные, ввел Джон фон Нейман.

Суть идеи заключается в том, что данные и код являются рекурсивными понятиями, т.к. код – это всегда данные для некоторой исполнительной машины. Это означает, что настройка любой программной системы и взаимодействие с ней всегда представляет собой ее программирование как некоторой абстрактной машины. Вот основной *принцип подобия*, излагающий эту точку зрения:

Код и данные подобны в том смысле, что любой код представляет собой данные для некоторой программной или аппаратной исполнительной машины.

Хранение кода совместно с данными порождает очень много вопросов. Во-первых: вопросы безопасности. Ведь данные, по сути, безопасны. А код нет. Однако, код и данные взаимозаменяемы, как уже обсуждалось выше. И вирусы, распространяющиеся через документы Word, файлы Bmp и Gif, вызывают много головной боли. В задаче решения таких проблем на первый план выходит *управляемый код* (managed code). Вопросы управляемого кода рассматриваются ниже в разделе «Архитектура современного ПО».

Во-вторых: вопросы зависимости. Сколько бы кода не было выгружено в файл данных, остается некоторая степень зависимости его от исполняющей машины и дополнительного программного кода. Это тоже следствие принципа подобия.

Все системы нужно программировать. Внесение возможности программирования системы каким-то путем, отличающимся от того, которым данная система создавалась, может отнять очень много ресурсов. В результате получается система, настройка которой предоставленными средствами обходится дороже, чем прямое программирование.

Еще раз, резюмируя вышесказанное, существуют две возможности: хранить в файле интерпретируемые данные, считая их всё же данными, и хранить в файле исполняемый код, зная, что это код. Преимущество хранения кода заключается в том, что снимается вопрос создания отдельного модуля, интерпретирующего данные в файле. Гораздо проще и правильнее воспользоваться стандартными существующими средствами исполнения управляемого кода, которые уже умеют работать с понятиями версий, безопасности, ограничений на исполнение и т.д.

В последнее время для всех крупных систем, от CAD/CAE до ERP, заметен дрейф в сторону применения общих средств для их программирования. Достоинство применения общих практик взамен специализированных, помимо всего прочего, усиливается одним важным психологическим аспектом, описанным ниже в разделе «Психологические аспекты разработки ПО».

#### Генотип[[10]](#footnote-10) и фенотип[[11]](#footnote-11)

В живых системах есть, по-видимому, очень интересный механизм реализации заложенных возможностей под конкретные условия. Он позволяет не вырастать слишком сильно, если этого не нужно. Быть более быстрым или более медлительным. И так далее, «по желанию заказчика».

Для программных систем подобное свойство важно в контексте используемой памяти и скорости работы. Баланс между кодом и данными, скоростью разворачивания и скоростью выполнения, определяются дизайном системы и представляют собой важные аспекты для обоих видов систем.

#### Поддержка состояния

Гомеостаз – механизм поддержки внутреннего состояния – очень важное свойство живых систем. Это также критично и для программ – система должна восстанавливаться от сбоев в стабильное состояние. Иначе непрерывная работа серверных приложений становится невозможной, в то время как мы круглосуточно звоним, заказываем билеты и узнаем прогноз погоды.

Традиционный подход к сохранению состояния системы нацелен только на сохранение данных. Это настолько сильно нарушает инкапсуляцию, что вопросы переносимости схем данных для объектных систем до сих пор не решены. Для чисто объектных баз данных смена схемы – катастрофа.

В живых системах мы не видим примеров, когда яйцо и зародыш хранятся отдельно. Видимо, в этом что-то есть. Видимо, неправильно вырывать данные из контекста объектов.

Изменение версии классов, затрагивающие схему данных объектов, можно сравнить с изменением генетического кода организма. Такое происходит при вирусной атаке (что нас не интересует) и при рождении нового поколения. Оба поколения[[12]](#footnote-12) либо могут жить вместе, как старая и новая версии платежного поручения в финансовом модуле ERP, либо старшее поколение уходит, передавая знания потомкам.

Для объектных баз данных, частным случаем которых являются и файлы проектов CAD систем, это означает хранение кода объектов рядом с данными. Учитывая современные отношения размеров кода к данным (допустим, 60 Кб кода к 20 Мб данных), это не кажется расточительным.

В виде программного кода распространяются сами программные продукты и их компоненты. Библиотеки компонент. А больше всего нас интересует распространение библиотек компонентов функциональных эквивалентных схем. Ниже в данном документе приводится более глубокое обоснование и пример использования такого подхода в приложении к ПА10.

#### Обратная совместимость

Живые системы поддерживают обратную совместимость. В нашем ДНК существует большое количество наследованного кода. Этот код позволяет нам при изменении условий «вспоминать» устаревшие гены. Обратная совместимость также важна и для программного обеспечения. Любая новая операционная система должна обладать полной программной совместимостью с большим набором прикладного ПО, иначе у нее нет почти никаких шансов стать первой ранее, чем за одно поколение разработчиков. Новая CAD или CAE, для успеха предприятия, должна понимать распространенные на момент релиза форматы данных. Выбор конкретного набора форматов стоит уже за экономикой, т.к. чем больше форматов мы поддерживаем, тем дольше и дороже обойдется нам разработка.

#### Стандартизация

Стандартизация - очень близкое к обратной совместимости свойство, важное для жизни в контексте размножения. Организму сложно размножаться, будучи нестандартным по отношению к другим особям. Размножение копированием самого себя обречено в чуть более длительной перспективе, т.к. неспособно адаптировать организм к новым условиям. Вероятно, стандартность даже важнее эффективности.

Стандартизация крайне важна и для программных систем. Ровно в том же смысле. По сравнению с организмами, программы находятся просто в тепличных условиях. В животном мире, если ты нестандартный или в чем-то хуже, и при этом занимаешь ту же нишу, что и другие пять стандартных, тебя лишат жизни гораздо быстрее, чем на планете установят еще один миллион копий Windows.

Вопрос о стандартности, вообще говоря, очень сложный. Следует помнить о стандартных интерфейсах взаимодействия, о соглашениях. Стандартные реализации при этом мало интересны. Подмена понятий стандартного подхода и стандартного (но ошибочного или ненадежного) *решения*, может дорого обойтись команде разработчиков.

### Историческая справка

#### Алан Кей

Одним из первых на живые системы в применении к программированию обратил внимание Алан Кей (Alan Key), создатель концепции объектно-ориентированного программирования и языка программирования Smalltalk.

Еще до поступления в 1966 году в аспирантуру университета штата Колорадо по теме «Математика и молекулярная биология», Алан отслужил на тренировочной базе Рандольф (Randolph), где работал с компьютером Burroughs 220[[13]](#footnote-13). Эта история по-разному перевирается различными изданиями. Здесь приводится версия журнала PC Magazine и самого Алана, которая кажется наиболее правдоподобной.

В 50-х не существовало единых стандартов на форматы документов. Еще до Алана Кея на базе Рандольф работал неизвестный программист, который создал под Burroughs 220 некоторое количество полезного программного обеспечения. Получаемые данные записывались на бобины с магнитной лентой. К сожалению, они не имели никакого смысла для компьютеров Burroughs других тренировочных баз, поскольку на них не было того уникального ПО. Вместо того чтобы распространить свои программы на другие машины, неизвестный гений нашел другое решение: он стал записывать на бобины данные в виде программ, которые содержали и данные, и алгоритмы обработки. Может показаться, что он придумал себе лишнюю работу.

В действительности, им было использовано очень прогрессивное решение, которое обладает следующими преимуществами:

1. Данные полностью переносимы в пределах определенной аппаратной платформы.
2. Изменения в формате данных не затрагивают существующие бобины. Данные по-прежнему доступны, и для этого не требуется дополнительного программирования.

Сущность преимуществ остается непонятной, пока мы смотрим на программное обеспечение, как на конструирование. Ну что нам стоит поддерживать несколько форматов наших же данных? По-видимому, Кей в то время также не оценил подхода. По долгу службы, в 1961 году, ему пришлось поддерживать имеющееся решение, и он слегка изменил его. Теперь на все машины Burroughs все же ставилась небольшая программа, предоставляющая пользовательский интерфейс для загрузки *модулей*, как он назвал имеющийся гибрид данных и программ. И только работая над кандидатской диссертацией в 1966 году, он понял всю оригинальность идеи.

В понимании Алана Кея, модули, переименованные им в *объекты*, представляли собой почти физически отдельные сущности, объединяющие данные и код, и обменивающиеся асинхронно друг с другом посредством передачи сообщений. Самым близким аналогом таких объектов могут служить изолированные процессы в современной многопользовательской операционной системе (ОС). Объекты в нотации Кея могли бы совместно работать в распределенной сети. Для того времени это было слишком революционно.

Заметим, что способ общения модулей в ПА7 посредством вызова друг другом большого количества исполняемых модулей с передачей данных через программные потоки ввода/вывода, во многом соответствует представлениям Алана Кея. Вероятно, в случае ПА7 сказывается влияние Unix систем, но в любом случае суть идеи сохраняется.

Объектно-ориентированное программирование, придуманное Кеем, вводит два основных принципа. Первый – принцип рекурсивного дизайна:

The basic principal of recursive design is to make the parts have the same power as the whole.

Alan С. Key: *The Early History of Smalltalk*, 1993, ACM 0-89791-571-2/93/0004/0069

Основной принцип рекурсивного дизайна – стремится сделать так, чтобы части обладали возможностями целого.

Алан Кей, *«Ранняя история SmallTalk»*, 1993, Издания компьютерного сообщества, статья №0-89791-571-2/93/0004/0069 (далее [KEY01])

Принцип постулирует, что программный объект должен обладать возможностями целого компьютера. Объектный анализ вычислительной машины с такой точки зрения говорит, что компьютер состоит из компьютеров меньшего масштаба. Для программной системы это выражается в том, что вся система, также как и её функциональные части, доступна в виде объекта. Самоподобие определяет структуру системы в процессе ее разработки, а это есть дизайн.

Второй принцип определен следующим образом:

…everything we can describe can be represented by the recursive composition of a single kind of behavioral building block that hides its combination of state and process inside itself and can be dealt with only through the exchange of messages.

…все, что мы можем описать, может быть представлено в виде рекурсивной композиции единственного типа поведенческих блоков, которые скрывают в себе комбинацию состояния и выполняющего процесса, и работа с которыми может вестись только посредством обмена сообщениями.

[KEY01]

Этот принцип вводит очень важную абстракцию, позволяющую не только инкапсулировать сложность, но и строить повторно используемые объекты. В современном понимании второй принцип Кея реализуется в *компонентах*. Компоненты – это очень спорное понятие, не меньше чем само объектно-ориентированное программирование.

Интерфейс компонента – это видимая извне, публикуемая, характеристика. Поэтому использование компонентного подхода наряду с декларацией основных интерфейсов определяют архитектуру будущей системы.

Рассмотрим подробнее следствия из указанных принципов, какими они видятся на современном этапе развития технологий, и подчеркнем некоторые распространенные, в какой-то степени ошибочные, представления. Собственно, при более подробном изучении вопроса оба принципа выражают одну и ту же точку зрения.

Применение практики рекурсивного дизайна оказывает серьезное влияние на программный проект. Самым простым следствием оказывается тот факт, что приходится писать больше программного кода. Это происходит потому, что рекурсивно стыкуемые элементы обязаны иметь понятный и унифицированный интерфейс. Применение разных критериев качества к «внешним» и «внутренним» компонентам системы становится признаком плохого дизайна, поскольку теперь нет по отдельности «внешнего» и «внутреннего».

Вторым следствием является более согласованная структура системы. Очевидно, что абстракция самого верхнего уровня, выражаемая объектом «Система» (System), «Приложение» (Application) или «Окружение» (Environment), предоставляет наиболее полный и доступный для понимания программный интерфейс[[14]](#footnote-14). Для прикладного программиста он определяет «лицо» системы. Настолько же высокие требования теперь должны быть представлены и ко всем другим составляющим. Результатом этого оказывается структура, отдельные элементы которой более доступны для непосредственного программного манипулирования и анализа, чем при разработке закрытых модулей. Такую структуру проще понять и проще подвергнуть тестированию.

Объединение данных с кодом, который их обрабатывает, до сих пор вызывает много споров между сторонниками ООП и защитниками процедурного программирования. Ранние пользователи С++, перешедшие на него с Си или процедурного Паскаля, отмечали полезность ухода от глобальных переменных модулей и функций. Превращение структур в классы означало для них избавление от ошибок несогласованного изменения общих данных. Основные доводы противников классов и инкапсуляции, как с той курицей, сводятся к высказываниям, вроде «если мы с Гошей программируем на Си очень аккуратно и согласованно, то у нас всё получается быстрее и лучше, чем со всеми этими «правильными» классами и объектами».

Изучая психологию детей на основе работы Сеймура Паперта[[15]](#footnote-15), Кей пришел к выводу о большом значении *интерактивной среды* программирования, которая бы обеспечивала возможность непосредственного взаимодействия с объектами программируемой системы. Созданная им впоследствии среда для Smalltalk позволяла программировать систему «в горячем режиме», т.е. добавлять и изменять код, не уничтожая существующие в системе объекты и потоки исполнения. Стоит отметить, что такая возможность была добавлена корпорацией Microsoft в новую версию их невероятно популярного продукта Visual Studio – MS Visual Studio 2005. Через 25 лет после Smalltalk-80.

В данный момент Алан Кей не считается авторитетом в среде объектно-ориентированного программирования. Он работает на компанию Walt Disney и разрабатывает новые виды аттракционов для детей и взрослых.

#### Бертран Мейер[[16]](#footnote-16)

Современные идеологи объектно-ориентированного программирования вводят несколько другие принципы и идеи. Наиболее известны из них двое – Грэди Буч и Бертран Мейер. Обратимся к Мейеру, т.к. книга у него толще и мысли в ней изложены в более сжатом виде. Цитировать Буча сложнее, поскольку законченная его цитата – это всегда полстраницы.

Довольно большое количество специалистов, особенно в России, говорят о том, что они хорошо понимают и постоянно используют объектно-ориентированный подход. В книге Мейера Object-Oriented Software Construction (Создание объектно-ориентированного программного обеспечения) слова misunderstand/misunderstood[[17]](#footnote-17) встречаются 14 раз.

Основным мотивом книги служит беспокойство о качестве программного обеспечения. Как и в случае многих других технических систем, качество подразумевает *корректность*, *надежность* и *эффективность*; плюс специфические для ПО *повторное использование* и *переносимость*.

По Мейеру всё это достигается простотой и децентрализацией частей:

The object-oriented method is, before anything else, a system architecture method which helps designers produce systems whose structure remains both simple (even for large systems) and decentralized.

Bertrand Meyer: Object-Oriented Software Construction, 2nd Edition, Prentice Hall, 1997

Объектно-ориентированный метод является, кроме всего прочего, методом создания архитектуры системы, который позволяет разработчикам получать системы, структура которой оказывается одновременно простой (даже для больших систем) и децентрализованной.

Бертран Мейер, «Создание объектно-ориентированного программного обеспечения, 2е издание», Prentice Hall, 1997 (далее [MYR01])

Децентрализация – очень важный принцип.

Decentralization: the more autonomous the modules, the higher the likelihood that a simple change will affect just one module, or a small number of modules, rather than triggering off a chain reaction of changes over the whole system.

Децентрализация: чем более автономными будут модули, тем больше вероятность, что простое изменение окажет влияние всего на один модуль или небольшое количество модулей, чем запустит цепочку изменений по всей системе.

[MYR01]

Децентрализация как раз означает, что нет какого-то выделенного «ядра» системы. Существуют только небольшие (в целях простоты) и максимально независимые (в целях децентрализации) модули. Этими модулями могут быть как классы, так и некоторые небольшие сообщества классов.

Фактически, присутствие централизации означает, что кто-то слишком много на себя берет. В физических или социальных системах это оправдывается большим неравенством возможностей. В программных системах возможно выделение сервера, опять же из-за влияния физических ограничений. Однако модули одной программы на одной машине чаще равноправны по своим возможностям – они используют один и тот же процессор, - и централизация себя не оправдывает. Выделение ядер и крупных функциональных модулей является, скорее, следствием психологических факторов; данный вопрос рассмотрен в разделе «Психологические аспекты разработки ПО».

Сам Мейер отмечает, что:

…I do not see the object-oriented method as a mere fad; I think it is not trivial…I know it works; and I believe it is not only different from but even, to a certain extent, incompatible with the techniques that most people still use today;

…я ничуть не вижу объектно-ориентированный метод странным; я думаю, он нетривиален… я знаю, что он работает; и я верю, что он не только отличается, но в некотором смысле несовместим с техниками, которые люди сейчас используют;

[MYR01]

Мейер также говорит о том, что принцип сокрытия информации (information hiding) неверно понимается ввиду своего названия. Согласно Мейеру, инкапсуляция и сокрытие информации – это не вопрос безопасности кода или данных (как в случае с глобальными переменными), а эффективный способ борьбы со сложностью. Инкапсуляция не защищает данные объекта от клиентов, но защищает пользователей от внутренней сложности объекта.

Наличие правильной среды программирования пропагандируется Мейером также активно, как и Аланом Кеем. По его мнению, правильная среда должна быть открытой, быстро реагировать на изменения, позволять активно исследовать структуру создаваемой системы и, что немаловажно, предоставлять программисту развитую стандартную библиотеку готовых компонентов.

В настоящее время Бертран Мейер преподает в швейцарском университете ETH[[18]](#footnote-18) в Цюрихе на кафедре программного обеспечения.

## Психологические аспекты разработки ПО

Очень часто можно слышать от разработчиков, что ОО программы непонятны для изучения, отладки и модификации. Фактически, это полностью противоречит тому, что декларируется в качестве достоинств объектно-ориентированного подхода.

Вопрос о сложности любой искусственной модели можно ставить с точки зрения двух критериев оценки: математическая вычислительная сложность, выражаемая временем работы абстрактной идеальной вычислительной машины, не нарушающей законов физики, и субъективная семантическая сложность, выражаемая в скорости преобразования используемых ею абстракций в некоторую внутреннюю воображаемую модель конкретного человека, и обратно.

В такой постановке мы имеем проблему приоритетов между двумя независимыми и возможно противоположными критериями. Очевидно, языки и парадигмы программирования используются для уменьшения разрыва между двумя оценками, обеспечивая высокую понятность исходного кода и высокую эффективность откомпилированной программы. Тема языков программирования рассматривается в разделе «Специализированные языки», в данном же разделе речь идет только об адекватности различных парадигм, реализуемых данными языками.

### Поток исполнения

Поток исполнения - одно из первых понятий, с которым возникают трудности при программировании вычислительных машин. На практике, очень незначительное число учебников по программированию вообще вводят это понятие в явном виде, несмотря на то, что оно неявно присутствует во всех программах начального курса. Вместо этого используют понятие *алгоритма*.

Существуют веские основания считать декларативный способ описания задачи более естественным для человеческого понимания, чем императивный. Человек строит ментальные модели и большая их часть императивна. Понимание *процесса*, стоящего за тем или иным явлением, относится уже больше к экспертной области, чем к области общего знания.

Преподавание информатики почти всегда преподносится как особый раздел математики, требующий именно математических способностей. Учитывая, что математические знания часто преподносятся именно как императивные, первое знакомство с программированием сопряжено с рядом распространенных ошибок и сложностей.

Несмотря на императивный характер процесса решения математических задач, постановка условий и отдельные этапы решения этих задач часто представлены в виде декларативных утверждений. Поэтому возмущение по поводу использования знака «=» в качестве процедуры присваивания, т.е. непонимание записей «I = I + 1», а также сложности с программированием циклов, непременно сопровождают первые этапы обучения.

Постепенно человек научается понимать и интерпретировать последовательный характер императивных программ. При этом программа воспринимается как набор операций, который всегда исполняется пошагово от первой и до последней строки, с несколькими циклами ветвлений внутри.

Практика объектно-ориентированного программирования использует гораздо больше декларативных элементов, чем при использовании, скажем, BASIC или даже PASCAL. Поэтому человеку, уже переученному с выражений на алгоритмы, необходимо снова переучиваться под классы. В процессе переучивания всегда наступает момент, когда в попытке удержать процедурный контроль над системой приходится «все время держать в уме все эти классы». В итоге, вместо использования information hiding для уменьшения необходимых знаний об используемых объектах, программист какое-то время держит в голове одновременно и публичный интерфейс, и внутреннюю структуру объектов.

Еще одной проблемой становится высокая сегментация кода. Большинство методов классов содержат теперь от 1 до 5 строк. Поток исполнения, выраженный в последовательных строках чисто императивной программы, теперь теряется в бесконечных вложенных вызовах методов объектов. Вот типичная реакция на хорошо децентрализованную ОО программу:

Непонятно что к чему – нет картинки. Где программа? Нужно чтобы видно было: так, так и так. Все должно быть в одном месте. Одно за другим, и все понятно.

Олег Третьяк, программный архитектор компании «Организационно-технологические решения» (цитата 2003 года)

Для человека, незнакомого с программированием, поток исполнения не имеет никакого значения, а ориентация на декларативное программирование гораздо более сильная. Поэтому написание нескольких строк условий и равенств, или задание настраиваемых значений (свойств объектов), не вызывает практически никаких сложностей.

Да что там писать, всё ж понятно.

Максим Матосов, студент МГТУ им. Баумана, кафедра РК6, о создании элементов механической подсистемы в ПА10 (цитата 2004 года)

### Элементы правильного дизайна

Как было сказано во введении, в основе любой деятельности лежит стремление к «правильности». В этом смысле, существует еще одно определение дизайна, которое привлекает своей лаконичностью:

Design is making things right.

Ralph Kaplan (1999): citation from *Design Research and New Learning* by Richard Buchanan, Design Issues: Volume 17, 2004

Дизайн означает делать вещи правильно.

Ральф Каплан (изречение 1999 года), цитата по *«Исследования дизайна и новое учение о нем»* Ричарда Буханана, Публикации о дизайне, том 17, 2004.

Здесь есть и процесс, и результат, и смысл происходящего. Но, конечно же, его нельзя ставить первым во введении.

В разное время правильными будут считаться разные вещи. В данный момент, в той или иной степени, признаком хорошего дизайна считается объектно-ориентированная технология. В этой связи существует одно негативное высказывание в сторону ООП со стороны приверженцев процедурного программирования: «ООП узурпировало понятие правильности». То же говорит и Бертранд Мейер:

“Object-oriented” is the latest **in** term, complementing and in many cases replacing “structured” as the high-tech version of “good”.

«Объектно-ориентированный» - последний интересный термин, во многих случаях заменивший высокотехнологичную версию «блага».

[MYR01]

La perfection est atteinte non quand il ne reste rien а ajouter, mais quand il ne reste rien а enlever.

Antoine de Saint Exupery.

Совершенство достигается не тогда, когда уже нечего больше добавить, а тогда, когда уже нечего больше отнять.

Антуан де Сент-Экзюпери.

The bad ideas and the complicated ones (they are often the same) often appear first; it takes time for the simple and the elegant to take over.

...[T]he single biggest enemy of reliablity (and perhaps software quality in general) is complexity.

Напрашиваются простые аналогии:

* Две сотни букв – три десятка букв;
* Двенадцатеричная система счисления – двоичная система;
* Сорок пар ног – две ноги;

Даже размер кода ДНК и количество генов не определяют сложности организма.

Лучший способ добиться эффективности - это создать ясный и простой проект. Только такой проект может остаться относительно устойчивым на весь период развития и послужить основой для настройки системы с целью повышения производительности. Здесь важно избежать "гаргантюализма", который является проклятием больших проектов. Слишком часто люди добавляют определенные возможности системы "на всякий случай", удваивая, учетверяя размер выполняемой программы ради завитушек. Еще хуже то, что такие усложненные системы трудно поддаются анализу, а по этому трудно отличить избыточные накладные расходы от необходимых и провести анализ и оптимизации на общем уровне.

Оптимизация должна быть результатом анализа и оценки производительности системы, а не произвольным манипулированием с программным кодом, причем это особенно справедливо для больших систем, где интуиция разработчика или программиста не может служить надежным указателем в вопросах эффективности.

Важно избегать по сути неэффективных конструкций, а так же таких конструкций, которые можно довести до приемлемого уровня выполнения, только затратив массу времени и усилий. По этой же причине важно свести к минимуму использование непереносимых по своей сути конструкций и средств, поскольку их наличие препятствует работе системы на других машинах (менее мощных, менее дорогих).

## Специализированные языки

Данный раздел посвящен вопросам специализированных языков программирования в CAD/CAE-системах: что такое специализированные языки, как они используются и где. Вопросы языков имеют непосредственное отношение к архитектуре системы, поскольку определяют правила разработки программных расширений.

### Введение

На заре компьютерной индустрии создание нового языка программирования было очень популярным занятием. Собственный язык считался задачей из ряда «посади дерево». Билл Киннерслей ведет список всех известных языков программирования на своей домашней страничке в Интернете[[19]](#footnote-19). На момент 2005 года на ней представлено около 2500 языков. И хотя там есть такие экзотические языки как:

* **473L Query.** Язык запросов для военной базы Air Force 473L, похожий на английский язык.

А технология COM[[20]](#footnote-20) занесена в языки, на ней не найти таких языков как:

* **EZPL** is EZ Programming Language**.** Язык программирования принтеров EZ.
* **BPEL** is Business Process Execution Language. Ориентированный на XML язык описания бизнес логики от IBM.

Собственно, различные индустриальные языки для принтеров, задания формата документов, диаграмм и т.д. занимают собой основную нишу специализированных языков.

Как можно определить специализированный язык? Самое первое интуитивное понимание вопроса говорит о том, что системные языки программируют вычислительные машины, а специальные – программируют специальные вычислительные машины или другие сложные программы. Языки общего назначения содержат весь доступный вычислительный базис, а специализированные, скажем так, не всегда Тьюринг эквивалентные[[21]](#footnote-21) (например, с помощью HTML нельзя даже складывать числа – это язык разметки).

Основными недостатками существующих языков общего назначения в применении к какой-то узкой области можно считать:

* Синтаксическую избыточность;
* Семантическую сложность;

Синтаксическая избыточность связана с широтой возможностей языков общего назначения и особенностями трансляции. Точки с запятой, двоеточия и фигурные скобки подчас затрудняют чтение и понимание кода программы, особенно для непрограммистов.

Борьбу со сложностью можно успешно вести, снижая количество объектов, которыми приходится манипулировать одновременно. Поэтому для разработчика почти всегда более короткая программа является субъективно более понятной и простой. Хотя это и не всегда так. Специализированные языки снижают сложность программирования прикладных технических задач, опуская понятия выделения памяти, указателей и прочие признаки машинной природы языков программирования.

Специализированные языки пытаются бороться и с тем, и с другим, предлагая инженеру наиболее компактную и понятную запись в рамках его проблемной области.

### Цена вопроса

Применение специализированного языка по определению должно быть более эффективным в известных прикладных задачах. Рассматривая вопрос создания специализированного языка, нужно дать ответ на следующие вопросы:

1. Стоимость разработки и стандартизации языка;
2. Стоимость поддержки и развития;
3. Эффект от применения.

Очевидно, что разработка имеет смысл в том случае, когда пункт 3 перекрывает пункты 1 и 2. Однако факторы, вносящие свой вклад в указанные пункты, немного более тонкие, чем следует из названия.

#### Стоимость разработки

С некоторого времени языки программирования стали цениться не столько за особенности синтаксических конструкций, сколько за мощность стандартной библиотеки, интеллектуальную среду исполнения (виртуальную машину, ядро) и богатство средств разработки. По этой причине написание даже очень хорошего, но одинокого, компилятора не способно привлечь к языку серьезного внимания. Как минимум, компилятор должен уметь интегрироваться с какой-нибудь популярной средой разработки.

Набором из строк, дат и даже работой с базой данных тоже никого не удивишь. Язык не будет применим без серьезной стандартной библиотеки. Даже если это специализированный язык. В то же время, конструкции нового языка должны манипулировать объектами предметной области, составляющих еще более серьезную часть стандарта.

Также существует тенденция, согласно которой появление языков, предлагающих новые синтаксические конструкции для известных структур, будет встречать такое же непонимание, как изобретение альтернативных терминов и нотаций. конструкции для известных структур, будет встречаться ем появлем тами предметной области, составляющих серьезную часть стандар Должна потребоваться очень серьезная аргументация в защиту того, что вместо знаков “//” или “ ’ ” для обозначения комментария используются какие-то другие символы.

#### Стоимость поддержки

За стоимостью поддержки языка кроется непрерывная гонка за новыми тенденциями в индустрии программного обеспечения. Объектно-ориентированное программирование, аспектное программирование, декларативные элементы, шаблоны и всё, что еще может появиться в будущем и оказать влияние на дизайн ПО. Даже язык разметки, каким является язык описания схемы, может легко оказаться под влиянием новых веяний теории программирования. История осложняется тем, что за собой приходится нести груз собственной нестандартной «стандартной библиотеки» и среды исполнения.

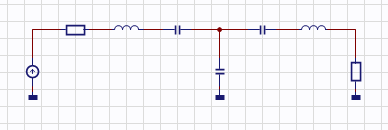
#### Эффект применения

Эффект от применения специализированного языка программирования во многом будет пропорционален количеству нового кода, создаваемого на планете вручную (а не с применением визуальных средств), и выигрыше утилит для автоматического анализа характеристик устройства/системы по ее исходному коду. Почти целиком этот эффект будет достигаться за счет нового синтаксиса. Потому как ни среда исполнения, ни стандартная библиотека не навязывают синтаксиса и наоборот.

### Языки существующих систем

Основным требованием к современной системе анализа проектных решений является наличие мощных и одновременно доступных возможностей по разработке и тестированию новых моделей технических устройств. В то же время, разработка моделей для MSC.ADAMS осуществляется на языке FORTRAN и C++, а в предыдущих версиях комплексов ПА (ПА7, ПА9) для этих целей применялись специальные языки на основе Си и Java.

В тексте ниже будут использоваться примеры программ, описывающих специальную тестовую схему. В качестве таковой выбрана схема фильтра высоких частот High-Q Filter (из теста T11 для PSPICE[[22]](#footnote-22)). Ниже приведено изображение этой схемы, как она выглядит в редакторе схем ПА10:



1. Схема High-Q Filter (из теста T11 для PSPICE)

#### PSPICE

В пакете PSPICE схема фильтра высоких частот будет описана в виде текстового файла с расширением «.CIR» и выглядеть следующим образом:

1. Код задания схемы High-Q Filter в PSPICE

VIN 1 0 1volt

R1 1 2 1ohm

L1 2 3 1001mH

C1 3 4 0.001uF

C2 3 5 1uF

C3 3 6 0.001uF

L2 6 7 999mH

R2 7 0 1ohm

.END

Приведенная программа состоит из набора последовательных команд, которые выглядят как декларативная запись топологии схемы. В действительности это процедурный язык, который содержит также операции “.PRINT”, “.PLOT” и т.д.

В данном примере, за исключением команды “.END”, используются только команды создания экземпляров элементов схемы. Команда создания имеет следующий формат:

NAME Node1 Node2 … NodeN Value

|  |  |  |  |
| --- | --- | --- | --- |
| где | NAME | - | имя элемента в виде <Заглавная буква, идентифицирующая тип + суффикс имени элемента>; |
| Node1 | - | номер одного из узлов, к которому будет подключен элемент; |
| Value | - | значение параметра элемента в виде <число + единица измерения>. |

В целом, код выглядит как очень компактная и полная запись топологии приведенной схемы.

#### ПА9

Комплекс ПА9 позволяет задавать расчетные схемы в виде текстовых файлов с расширением «.mm9». Код схемы T11 будет выглядеть следующим образом:

1. Код задания схемы High-Q Filter в ПА9

$MODEL: T11;

#NODES: n0, n1, n2, n3, n4, n5, n6, n7;

#TOP:

VIN: <V> n1 n0 = 1e0;

R1:<R> n1 n2 = 1e0;

L1:<L> n2 n3 = 1001e0;

C1:<C> n3 n4 = 1e-3;

C2:<C> n3 n5 = 1e0;

C3:<C> n3 n6 = 1e-3;

L2:<L> n6 n7 = 999e0;

R2:<R> n7 n1 = 1e0;

Соответствующее описание схемы для ПА7 выглядит почти точно так же. За исключением некоторых синтаксических особенностей, данный код очень похож на то, что мы видим в PSPICE.

#### ПА10

Полный пример кода на C#, реализующего необходимый набор операций для схемы фильтра высоких частот High-Q Filter, приведен в Приложении 2. Ниже приведен лишь небольшой отрывок кода, показывающий характер выполняемой работы:

1. Фрагмент кода сборки и решения эквивалентной схемы High-Q Filter на C#.

 // создание элементов.  
 BaseNode b1 = new BaseNode();  
 // соединение элементов с узлами  
 b1.ConnectedContacts.AddRange(uSource.InContact, c2.InContact, r2.OutContact);  
// подключение индикаторов  
 PotentialIndicator in3 = new PotentialIndicator(n3);  
 // Решение:  
 // создание экземпляра вычислительного компонента и настройка параметров расчета.  
 Pa10.Mathematics.DmanSolver solver = new Pa10.Mathematics.DmanSolver();     
 solver.Schema = Schema.Collect(b1);

// запуск на расчет  
 solver.Evaluate();

Также в целях наглядности ниже приведен фрагмент функционально того же кода на C++:

1. Фрагмент кода сборки и решения эквивалентной схемы High-Q Filter на C++.

 // создание элементов.  
 BaseNode\* b1 = new BaseNode();  
 // соединение элементов с узлами  
 b1->ConnectedContacts.AddRange(uSource->get\_InContact(), c2->get\_InContact(),

r2->get\_OutContact());  
// подключение индикаторов  
 PotentialIndicator\* in3 = new PotentialIndicator(n3);  
 // Решение:  
 // создание экземпляра вычислительного компонента и настройка параметров расчета.  
 Pa10.Mathematics.DmanSolver\* solver = new Pa10.Mathematics.DmanSolver();     
 solver->Schema = Schema.Collect(b1);

// запуск на расчет  
 solver->Evaluate();

Анализ кода показывает, что он в большей своей части является декларативным, а значит может быть записан в более наглядном и компактном виде на некотором декларативном языке. Например на XML. Полный пример создания (но не решения) ровно той же схемы на XML приведен в Приложении 3. Ниже также приведен небольшой отрывок данного примера:

1. Пример сборки схемы High-Q Filter на XML

<schema xmlns="http://www.pa10.org/xsdschemas/Schema.xsd">

<elements>

<BaseNode id="b1">

<connect element="uSource" contact="InContact" />

</BaseNode>

<PotentialIndicator id="in3" node="n3" />

</elements>

<solution>

<DmanSolver id="solver" Epsilon="0.01" EndTime="128" DeltaTimeMax="128.0"

DeltaTimeMin="128.0e-12" IntegrationMethod="AStableImplicitQuadroPrecise" />

</solution>

</schema>

Первый взгляд на код в приложениях может привести к мысли: «Что-то здесь не так! Слишком много программного кода для такой простой задачи». Код PSPICE для сборки схемы (без учета индикаторов и математического компонента) оказался более чем в 3 раза короче. «В три раза» звучит серьезно почти для любой задачи. Что же произошло? Ответ на этот вопрос дается в следующем параграфе.

#### Анализ кода PSPICE

Проанализируем код задания топологии фильтра высоких частот с целью выяснить природу различий между ним и кодом задания той же схемы на XML. Приведем код PSPICE еще раз:

VIN 1 0 1volt

R1 1 2 1ohm

L1 2 3 1001mH

C1 3 4 0.001uF

C2 3 5 1uF

C3 3 6 0.001uF

L2 6 7 999mH

R2 7 0 1ohm

.END

Строковая структура языка схем PSPICE, вероятнее всего, берет свое начало от языков для ввода с перфокарт. Вводя данные с перфокарт, вы имеете одну перфокарту на одну команду языка. При этом структура команды на перфокарте имеет следующий вид:

<метка> <идентификатор команды (обычно до 5 символов)> <параметр1>, <параметр2>…

Язык XML не различает строки и вводит сходное понятие *тега*. Переписав код из примера в виде тегов, мы получим следующий текст:

1. Код PSPICE, записанный в теговом стиле

<schema>

<VIN 1 0 1volt />

<R1 1 2 1ohm />

<L1 2 3 1001mH />

<C1 3 4 0.001uF />

<C2 3 5 1uF />

<C3 3 6 0.001uF />

<L2 6 7 999mH />

<R2 7 0 1ohm />

</schema>

В данном примере тег <schema> заменяет собой начало документа и команду «.END». Когда вы программируете на перфокартах, использование угловых и фигурных скобок сравнимо с безумием, каждый байт на счету. С удешевлением памяти и развитием средств ввода данных, на первое место выходят возможности анализа программного кода – и визуальные, и автоматические.

Рассматривая использование индексов узлов, мы находим еще одно часто используемое упрощение, которое заключается в объединении декларации переменной и ее использования. С точки зрения существующих практик программирования, декларация должны быть произведена явно. Соответственно, в нашем примере появятся предварительные определения всех используемых узлов:

1. Код PSPICE, записанный в теговом стиле, с декларацией узлов

<schema>

<N0/>

<N1/>

<N2/>

<N/>

<N4/>

<N5/>

<N6/>

<N7/>

<VIN N1 N0 1volt />

<R1 N1 N2 1ohm />

<L1 N2 N3 1001mH />

<C1 N3 N4 0.001uF />

<C2 N3 N5 1uF />

<C3 N3 N6 0.001uF />

<L2 N6 N7 999mH />

<R2 N7 N0 1ohm />

</schema>

Следующими на очереди стоят именованные параметры. Под именованными параметрами понимается запись операции передачи данных объекту в стиле, когда семантика значения привязывается не к позиции данных в списке параметров, а к некоторому публичному имени параметра. При именовании параметров их порядок перестает иметь значение.

Следует сказать несколько слов о практике использования наименований в современных языках программирования. Использование явно указываемых имен значительно упрощает анализ исходного кода. В то же время, пропорционально увеличивается семантическая зависимость модулей. Имя параметра – это публичный интерфейс метода объекта. Изменение имени параметра приводит к необходимости изменений во множестве участков зависимого кода, использующего вызов методов с данным параметром. Один из основных аргументов в защиту именованных параметров такой: с развитием системы скорее изменится композиция объектов и количество параметров методов, чем имена этих параметров. Код PSPICE с именованными параметрами приведен ниже.

1. Код PSPICE, записанный в теговом стиле, с декларацией узлов и именованными параметрами

<schema>

<N id= "N0"/>

<N id="N1"/>

<N id="N2"/>

<N id="N3"/>

<N id="N4"/>

<N id="N5"/>

<N id="N6"/>

<N id="N7"/>

<V id="VIN" in="N1" out="N0" value="1volt" />

<R id="R1" in="N1" out="N2" value="1ohm" />

<L id="L1" in="N2" out="N3" value="1001mH" />

<C id="C1" in="N3" out="N4" value="0.001uF" />

<C id="C2" in="N3" out="N5" value="1uF" />

<C id="C3" in="N3" out="N6" value="0.001uF" />

<L id="L2" in="N6" out="N7" value="999mH" />

<R id="R2" in="N7" out="N0" value="1ohm" />

</schema>

Такая запись уже достаточно сильно похожа на код в Приложении 3. В ней объединены данные и метаданные. Этот код сложен для ручного набора, но прост для визуального анализа и программного манипулирования. Его намного легче сгенерировать автоматическим генератором кода и легче понять сторонней программе.

#### Выводы

Код для ПА10 оказался чрезвычайно избыточным и неэкономным. Это ставит перед нами вопрос о целесообразности использования именно такой записи.

Ответ лежит на поверхности: такой код не обязательно создавать вручную. Применение визуальных редакторов моделей способно свести ручное редактирование кода к нулю. Таким образом, запись в XML в большей степени ориентирована на возможность загрузки и анализа ее сторонними системами.

Оценивая популярность пакета PSPICE и количество созданных для него моделей, можно поставить вопрос о создании транслятора схем .CIR в схемы или даже готовые компоненты для ПА10. Очевидно, что это несложная и выполнимая задача.

# Технологическая часть

## Архитектура современного ПО

Под современным программным обеспечением в данном разделе понимается только то тиражируемое промышленное программное обеспечение, разработка которого требует привлечения одновременного труда большого числа разработчиков и представляет собой крупный программный проект. Академические проекты и мелкое shareware[[23]](#footnote-23) обеспечение не учитываются.

За определение крупного проекта можно взять следующее высказывание Йири Саукапа[[24]](#footnote-24):

It is well known that most software cost is in maintenance, not in code development. In that sense, serious software is almost synonymous with maintainable software. Software is easy to maintain if it has clear internal organization, if it consists of modules that can be tested independently, and if it has logic which is easy to understand for a new programmer assigned to the project.

Juri Soukup: *“Taming C++”*, Addison Wesley, 1996

Известно, что большая часть стоимости программ заключается в поддержке, а не в написании кода. В этом смысле, серьезное программное обеспечение является полным синонимом поддерживаемого программного обеспечения. ПО легко поддерживать, если оно имеет ясную внутреннюю структуру, состоит из модулей, а ее логику легко понять для нескольких новых программистов, примкнувших к проекту.

Йири Саукап, *«Укрощение C++»*, Addison Wesley, 1996

Любая промышленная CAD или CAE имеет большие шансы быть «крупным проектом».

Практически всегда большие системы являются открытыми для расширений, будь это scripts, plug-ins или какие-то другие средства. Практически всегда существует необходимость интеграции с другими системами.

Архитектура современных систем обязательно содержит следующие элементы:

* Компоненты;
* XML;
* Распределенная работа.

Это современно, модно и в целом является признаком хорошего вкуса.

Раскрывая перечисленные пункты, о современной системе можно сказать, что она:

* Расширяется программами на любом промышленном языке и состоит из модулей, которые можно использовать в своих целях;
* Использует стандартные, открытые и самодокументированные форматы данных для обмена;
* Обеспечивает одновременную работу нескольких пользователей, эффективно использует ресурсы сети.

Маркетинговые документы практически любой известной системы, начиная с 2000 года, стали содержать пункт «оно использует XML; где-то там внутри есть XML».

Создание CAD/CAE систем, что вполне естественно, находится под сильным влиянием общих тенденций в индустрии программного обеспечения. Поэтому объектно-ориентированное программирование, компоненты и XML широко в них используются.

### Компонентная технология

Какие технологии и решения в действительности способны продлить срок жизни продукта? По всей видимости, нельзя полагаться на то, что на данном этапе развития технологий какой-то конкретный язык программирования или формат данных будет актуальным более 10 лет.

Примеры из математики и физики показывают нам, что какие-то соглашения могут быть достаточно постоянными. Нотация записи формул, некоторые способы решения, остаются неизменными по крайней мере последние 60 лет. При этом нельзя сказать, что в математике или физике мы видим меньше нововведений и изменений парадигмы.

Все дело в стандартизации. К определенному моменту становится понятным, что изменение взгляда на проблему, появление новых способов решения, не обязательно должны приводить к изменению нотации. Наоборот, математика, физика, химия, социология – стремятся унифицировать нотацию.

Можно предположить, что подобное должно произойти и с языками программирования, и с языками хранения и представления данных. Они могут быть максимально унифицированы и уже не подвергнутся частым изменениям, хотя лежащие в их основе технологии и будут меняться.

Посылкой для объединения языков в данный момент является компонентная технология. Сами по себе компоненты могут быть написаны на любом языке программирования и представляют собой основное звено интеграции разнородных программ и библиотек. Однако можно ожидать, что в сфере промышленного программирования крупных приложений, выживет и займет наибольшую нишу тот язык, который сможет наиболее полно и просто выразить компонентный подход.

Ниже дается определение компонента по замечательной книге Маркуса Вёльтера и других «Паттерны серверных компонентов», дающей обширное введение в тематику разработки компонентов для бизнес-приложений:

A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.

Clemens Szyperski: Component Software - Beyond Object-Oriented Programming, Addison-Wesley, 1999 (citation from Markus Völter, Alexander Schmid, Eberhard Wolff: *Server Component Patterns*, John Wiley & Sons, 2002)

Программный компонент – это сборочная единица с заранее оговоренными интерфейсами и только явно указанным контекстом зависимостей. Программный компонент может распространяться независимо, и предназначен для использования в сборке сторонними разработчиками.

Клеменс Сциперски, «Компонентное программное обеспечение – по ту сторону объектно-ориентированного программирования», Addison-Wesley, 1999 (цитата по Маркус Вёльтер, Александр Шмид, Эберхард Вольф, «Паттерны серверных компонентов», John Wiley & Sons, 2002)

Таким образом, компонент – это бинарная единица, не связанная напрямую с используемым языком программирования. Компонент может распространяться как независимая единица, соединяя в себе и код и необходимые данные (называемые ресурсами). Именно компоненты обещают заложить эру действительно широкого повторного использования программных модулей.

Есть существенное различие между программной библиотекой и компонентом (или библиотекой компонент). Суть в том, что компонент является в полном смысле независимой единицей. Это объектно-ориентированное понятие. Компонент сам управляет необходимой ему памятью и другими ресурсами, не требует «инициализации библиотеки» и публикует самодокументированный стандартный интерфейс взаимодействия. Вся функциональность по визуальному отображению данных, сохранению состояния на диск и прочее реализуется самим компонентом и доступна через методы любого экземпляра данного компонента. Для приверженцев процедурного программирования эта разница совсем не видна или кажется надуманной. В действительности разница значительна, т.к. обозначает собой серьезный сдвиг парадигмы в мышлении разработчика и сразу же проявляется при начале работы с профессиональными компонентами.

С приходом технологии .NET от Microsoft, в мире IT становится намного меньше экстремизма в отношении различных языков программирования. Теперь большинство специалистов, воспитанных на устаревших технологиях, могут немного расслабиться. Вот цитата из книги Маркуса Кнасмюллера «From COBOL to OOP»:

Taking an analogy from natural life, although the dinosaurs are extinct, a species called crossopterygian (a large group of fishes that have paired fins suggesting limbs) have been around 250 million years longer than dinosaurs and are still here. Therefore, no doubt COBOL programmers will also survive without going to the effort of making an evolutionary adjustment.

Markus Knasmüller: *“From COBOL to OOP”*, Morgan Kaufman Publishers, 2001



1. Коэлакант (примитивный crossopterygian)

Обращаясь к аналогии с биологической жизнью, несмотря на то, что динозавры вымерли, вид, называемый crossopterygian (большая группа рыб, которые имеют пару плавников – зачатков легких), существовали примерно за 250 миллионов лет до динозавров, и до сих пор существуют. Поэтому нет сомнений, что программисты на COBOL выживут без необходимости попыток эволюционных изменений.

Маркус Кнасмюллер, *«От COBOL к ООП»*, Morgan Kaufman Publishers, 2001

Компонентная технология способна продлить срок жизни реализаций определенных алгоритмов. К примеру, если у вас уже есть набор классов Zip, GZip, BZip2 в виде компонент, повторно создавать реализации указанных архиваторов уже нет нужды, даже если вы программируете на Perl или COBOL.

Стоит только немного задуматься над указанным применением компонентов, как встает очевидный вопрос: возможно ли повторное использование бинарного компонента при изменении аппаратной платформы и операционной системы? Ответ на него дает понятие управляемого кода.

### Управляемый код

Допустим, мы хотим воспользоваться программным компонентом, разработанным третьей стороной. Сам компонент доступен нам в виде нолей и единиц, загруженных из Интернет. Для успешного его применения мы должны получить ответы на следующие вопросы:

1. Формат данных файла компонента;
2. Способ загрузки и запуска кода компонента на разных программно-аппаратных архитектурах;
3. Безопасность:
   1. Подпись компонента производителем;
   2. Возможность наложить ограничения на права исполнения.

Допустим, что формат данных как-то зафиксирован и утвержден, а цифровую подпись можно проверить. Формат кода компонента также придется утверждать. Это означает, что необходимо зафиксировать набор и формат команд абстрактного унифицированного процессора, под который и будет компилироваться компонент с первоначального произвольного языка.

## Дизайн современного ПО

В том случае, когда исходные коды системы недоступны, о ее дизайне можно судить по доступным компонентам, предоставляющем какие-то публичные интерфейсы. Чаще всего это объект, реализующий интерфейс Environment или Application, который передается системой в главный метод (или конструктор) внешнего программного расширения системы. Вот список названий для нескольких систем:

* **Solid Edge** – SolidEdge.Application и SolidEdge.Document;
* **AutoCAD** - AutoCAD.Application и AutoCAD.Document;
* **MS Visual Studio** - ENV.DTE.

Из такого объекта становятся доступными коллекции всех загруженных документов, элементы этих документов, средства доступа к файлам и базам данных системы. Довольно часто экземпляр Application доступен в виде ActiveX компонента для сторонних программ.

К сожалению, для перехода от процедурного API[[25]](#footnote-25) к объектно-ориентированному нужно время. Первые реализации компонентной модели для C++ тяготели к избыточной сложности программирования. Ввиду наследованных особенностей готового программного кода и новизны подхода, первые реализации ООП API выглядят недостаточно профессионально.

В основном дают о себе знать процедурное прошлое системы и сложность реализации COM. Процедурное прошлое проявляется в функциях, принимающих сложные структуры, а сложность COM в малом числе методов, выполняющих несколько функций каждый, в зависимости от значения некоторого параметра, и отказ от применения пользовательских объектов по причине отсутствия управляемой памяти. К примеру, наряду с полностью объектным первым уровнем, можно встретить коллекции, по которым нужно пробегаться вызовом методов

NextChild(int skipCount, IUnknown\*\* element, int\* realSkipped),

и которые меняют внутреннее состояние этой коллекции. Еще чаще встречается обработка команды пользовательского интерфейса через эмулирующую полиморфизм командную функцию

ExecCommand(GUID commandId, int executionType),

которая содержит switch по коду команды, совсем как в процедурном Win23API.

В качестве примера, ниже приводятся три примера кода для Solid Edge по книге *“Solid Edge Programmers Guide”*, Unigraphics Solutions, Inc.

1. Код для перемещения двух фигур на Visual Basic

This program connects to a running instance of Solid Edge, places two parts, and repositions one of them.

'Declare the program variables.

Dim objApp as Object

Dim objOccurrences as Object

Dim objBox as Object

Dim objLid as Object

'Connect to a running instance of Solid Edge.

Set objApp = GetObject(,"SolidEdge.Application")

'Open an existing document.

Call objApp.Documents.Open("c:\My Documents\Drawing Files\file.asm")

'Access the Occurrences collection.

Set objOccurrences = objapp.ActiveDocument.Occurrences

'Place two grounded parts.

Set objBox = objOccurrences.AddByFilename("C:\My Documents\Drawing Files\Box.par")

Set objLid = objOccurrences.AddByFilename("C:\My Documents\Drawing Files\Lid.par")

'Move the lid to the correct location.

Call objLid.Move(0, 0, .055)

1. Заголовки методов интерфейса Add-in на C++

HRESULT OnCommand( long nCmdID )

HRESULT OnCommandUpdateUI( long nCmdID, long\* pdwCmdFlags, BSTR

Menutext, long \*nIDBitmap)

WindowProc( IDispatch\* pDoc, IDispatch pView, UINT nMsg, WPARAM wParam,

LPARAM lParam, LRESULT \*lResult )

1. Заголовок метода добавления новой страницы Edgebar на C++

HRESULT AddPage( IDispatch \*theDocument, long nInstanceHandle,

long nBitmapID, BSTR strTooltip, long nOption, long \*hWndPage)

Отказ от применения объектного паттерна Command в случае Add-in обусловлен, по всей видимости, именно отсутствием сборки мусора и возможными ошибками с памятью под C++ (создание объекта на стеке и пр.).

Чудовищно. Просто чудовищно.

Евгений Алиев об объектной модели пользовательского интерфейса *Solid Edge V17*, цитата 2005 года

# Практическая часть

## Архитектура комплекса ПА10

Данный раздел посвящен архитектурным особенностям комплекса ПА10 и обоснованием причин, поудивших принять именно такие проектные решения. ПА10 изначально позиционировалась как удобное и мощное средство для моделирования различных физических систем, сфера применения которого не должна ограничиваться только этапом создания и анализа модели устройства.

### Введение

Архитектурно, комплекс ПА10 представляет собой набор слабосвязанных программных компонент, над которыми в качестве фасада надстроено оконное приложение дизайнера схем.

Модели ПА10 представляют собой программные сборки компонентов под Microsoft .NET. В таких сборках содержатся и исполняемый код элементов, и оснастка для редактирования их экземпляров в дизайнере, и все необходимые ресурсы компонентов, в том числе строковые для любого количества естественных языков планеты.

Технология Microsoft.NET позволяет получить в результате функциональной композиции не просто абстрактные модули или библиотеки, а бинарные файлы сборок, содержащие в себе код независимых свободно распространяемых компонентов. В частности это означает, что станицу редактирования схем ПА10 можно действительно легко внедрить в любое другое .NET приложение.

Это несколько отличается от того, что можно найти среди CAD/CAE систем в настоящее время. Действительно, пакет AutoCAD также позволяет встраивать основное окно редактора в другие приложения в качестве ActiveX компонента. Однако на этом интеграция заканчивается. Вы должны иметь полный установленный пакет AutoCAD на своей машине, и на всех машинах пользователей, и это всегда будет целиком AutoCAD, а не какая-то его функциональная часть. Очевидно, что Autodesk позиционирует и поставляет свой пакет как цельный продукт, и поэтому не вводит полумер.

В то же время, сфера применения CAD/CAE систем не ограничивается редактированием схем на рабочем месте инженера. Прогресс не стоит на месте, и на наших глазах рождаются десятки новых видов устройств, занимающих те ниши, в которые PC не смог проникнуть в «большом» варианте. Растет потребность в интеллектуальных свойствах приложений. Производительность систем растем, а потому редакторы трехмерной анимации, компьютерные игры и интерактивные презентации уже могут позволить себе «честные» вычисления физических параметров модели.

Большим CAD системам уже пришлось однажды спуститься с Mainframe[[26]](#footnote-26) на PC. Настало время расстаться и с монолитной архитектурой.

Благодаря .NET, математический модуль ПА10, базовые элементы ФЭС, редактор схем, сборщик новых компонентов ФЭС и т.д. могут быть использованы независимо друг от друга.

### Узлы декомпозиции

Разбиение на компоненты и их взаимодействующие группы удобно вести по функциональному признаку. Существует несколько задач и разных областей, которые необходимо решить для того, чтобы получить на выходе работающий продукт. Эти задачи можно условно назвать модулями:

1. Математический модуль;
2. Объектная модель схем;
3. Визуальное редактирование;
4. Генератор кода компонента по известной схеме;
5. Сохранение и загрузка схем и компонентов;
6. Анализ данных;

Каждая их этих задач распадается на большое количество подзадач, реализуемых собственным семейством компонентов пакета.

#### Математический модуль

Математический модуль комплекса ПА10 состоит из двух частей:

1. Неуправляемая библиотека математического метода (используется DMAN), оптимизированная под конкретный процессор;
2. Набор управляемых компонентов для сборки матрицы в соответствии с требованиями вычислительных компонентов, участвующих в решении, конвертирования и пересылки данных, а также разборки матрицы с результатами.

Такое разделение преследует две цели:

1. Повысить производительность матричных вычислений;
2. Оформить взаимодействие с матрицами высокоуровневым объектно-ориентированным способом.

Общеизвестно, что объектные коллекции и управляемые массивы (не выровненные по границам страницы, с проверкой выхода за пределы массива и т.д.) не лучшим образом обеспечивают скорость инженерных расчетов. И хотя основная скорость вычислений достигается за счет алгоритмов, а не за счет платформы, в том случае, когда алгоритмы хорошо известны и могут быть реализованы наиболее оптимальным способом, лучше реализовывать их так же максимально близко к оптимальным машинным кодам ЭВМ.

Наиболее механичной работой при использовании готовой математической библиотеки для проведения практических расчетов является заполнение коэффициентов матрицы полученной системы уравнений[[27]](#footnote-27). Объектные компоненты и интерфейсы призваны облегчить эту задачу. Помимо прочего, абстракция от матриц позволяет использовать несколько различных методов решения систем. Фактически, в математическом модуле ПА10 разработчик эквивалентных элементов не имеет представления об индексах коэффициентов в матрице.

Ниже приводится пример кода вычислительной функции для элемента «Емкость»:

1. Вычислительный метод для емкости

/// <summary>

/// Выполняет вычисления для следующего шага решения.

/// </summary>

/// <param name="e">Компонент доспута к данным матрицы.</param>

protected override void Solve(DmanEvaluationDataAcessor e)

{

// I - U \* G = 0

e.EqutationValues[this.IVariable] = e.Values[this.IVariable] –

this.conductance \* e.Values[this.UVariable];

e.Jacobi[this.IVariable, this.IVariable] = 1;

e.Jacobi[this.IVariable, this.UVariable] = -this.conductance;

// U - (phi\_out - phi\_in) = 0

e.EqutationValues[this.UVariable] = e.Values[this.UVariable]

+ e.Values[this.OutContact.Node.PhiVariable]

- e.Values[this.InContact.Node.PhiVariable];

e.Jacobi[this.UVariable, this.UVariable] = 1.0D;

e.Jacobi[this.UVariable, this.OutContact.Node.PhiVariable] = 1.0D;

e.Jacobi[this.UVariable, this.InContact.Node.PhiVariable] = -1.0D;

}

В данном примере:

|  |  |
| --- | --- |
| e.EqutationValues | Предоставляет доступ к столбцу значений алгебраических уравнений. Пользователь не может получить доступ к значению произвольной строки, если не имеет доступа к компоненту, определяющему эту строку. |
| this.IVariable | Экземпляр переменной *потока*, принадлежащей данной емкости. Каждый компонент, участвующий в вычислениях, определяет свой набор переменных и публикует их для внешних математических компонентов, к которым принадлежат также ассемблер («сборщик») матрицы системы уравнений. |
| this.UVariable | Экземпляр переменной *потенциала*, принадлежащей данной емкости. |
| e.Values | Предоставляет доступ к значениям переменных на предыдущем шаге. Опять же, не зная переменную нельзя получить ее значение (поскольку числовые индексы в методах доступа не фигурируют). |
| e.Jacobi | Доступ к ячейкам матрицы Якоби. |

Перед началом вычислений, основной математический компонент, реализующий интерфейс Pa10.Mathematics.ISolver, собирает данные обо всех участвующих в вычислении компонентах. Компоненты должны реализовывать интерфейс Pa10.Mathematics.IComponent:

1. Описание интерфейса Pa10.Mathematics.IComponent

/// <summary>

/// Интерфейс вычислительного компонента.

/// Компонент – элементарная единица вычислительного процесса. Каждый компонент

/// описывает часть вычислительной задачи и символизирует собой одно или несколько

/// уравнений в общей системе ДАУ.

/// Заметим, что, на самом деле, компонент далеко не всегда знает в каких вычислениях

/// ему приходится участвовать. В большом числе случаев он лишь несет описательную

/// информацию, такую как свойства или, например, дифференциальное уравнение в

/// текстовом виде. В компоненте безусловно могут производиться какие-то вычисления,

/// но опять же, не всегда результаты этих вычислений могут быть сразу же, без

/// дополнительной обработки, применены при решении системы уравнений.

/// Для перехода от данных и вычислений компонента к непосредственному заполнению

/// матрицы Якоби и вектора значений функций (что собственно и требуется для решения

/// системы ДАУ низкоуровневым математическим ядром) введен интерфейс

/// <see cref="IComponentController"/> . Объекты, реализующие этот интерфейс,

/// производят необходимые действия для заполнения матриц и другие сопутствующие

/// вычисления.

/// </summary>

public interface IComponent

{

/// <summary>

/// Возвращает контроллер данного компонента.

/// </summary>

IComponentController Controller

{

get;

}

}

Чаще всего классы элементов функциональных схем реализуют также и интерфейс Pa10.Mathematics.IComponentController:

1. Описание интерфейса Pa10.Mathematics.IComponentController

/// <summary>

/// Интерфейс контроллера, предоставляющего мат. информацию о компоненте:

/// колличество переменных, расчет значений уравнений и частных производных.

/// </summary>

public interface IComponentController

{

/// <summary>

/// Возвращает массив внутренних переменных компонента.

/// </summary>

VariableReadonlyCollection Variables

{

get;

}

/// <summary>

/// Выполняет рассчет состояния компонента в текущий момент времени.

/// </summary>

/// <param name="evaluationData">Предоставляет доступ к различным данным для

/// рассчета. Подробнее смотри <see cref="IEvaluationDataAcessor"/></param>

void Solve(IEvaluationDataAcessor evaluationData);

/// <summary>

/// Выполняет инициализацию компонента в начале рассчета.

/// </summary>

/// <param name="initData">Предоставляет доступ к различным данным для

/// инициализации.

/// Подробнее смотри <see cref="IEvaluationDataAcessor"/></param>

void Init(IEvaluationDataAcessor initData);

/// <summary>

/// Выполянет обработку результата.

/// </summary>

/// <param name="resultData">Доступ к различным данным для обработки

/// результата. Подробнее смотри <see cref="IEvaluationDataAcessor"/></param>

void ResultsProcessing(IEvaluationDataAcessor resultData);

/// <summary>

/// Возвращает имя указанной переменной без постфикса компонента.

/// </summary>

/// <param name="variable">Переменная, имя которой без постфикса компонента

/// необходимо вернуть.</param>

/// <returns>Имя указанной переменной без постфикса компонента.</returns>

string GetVariableName(IVariable variable);

/// <summary>

/// Возвращает строковый массив уравнений, задаваемых данным контроллером.

/// </summary>

/// <param name="namingService">Служба наименования переменных, которая

/// позволяет получить строковые названия интересуемых переменных.</param>

/// <returns>Строковый массив уравнений, задаваемых данным

/// контроллером.</returns>

string[] GetEquations(VariableNamingService namingService);

}

Последние два метода, GetVariableName и GetEquations, позволяют построить реконструкцию символьной записи системы уравнений по ее объектному представлению.

#### Объектная модель эквивалентных функциональных схем в ПА10

Объектная модель функциональных эквивалентных схем в ПА10 строится вокруг класса Pa10.Modeling.Schema. Схема содержит в себе базовый узел и коллекцию всех входящих в нее элементов.

1. Описание внешнего интерфейса класса схемы

/// <summary>

/// Представляет собой функционально-эквивалентную схему, собранную из

/// различных элементов.

/// </summary>

public class Schema

{

/// <summary>

/// Создает экземпляр Schema.

/// </summary>

public Schema() {…}

/// <summary>

/// Возвращает базовый узел данной схемы.

/// </summary>

public BaseNode BaseNode [get;]

/// <summary>

/// Возвращает коллекцию компонентов, составляющих данную схему.

/// </summary>

public Pa10.Mathematics.ComponentCollection Components [get;]

/// <summary>

/// Выполняет поиск и сборку компонентов в коллекцию данной схемы.

/// </summary>

/// <param name="component">Компонент, из которого необходимо начать

/// поиск связных цепей для включения в схему.</param>

public void Collect(Pa10.Mathematics.IComponent component) {…}

}

Большинство потребностей инженера по созданию и анализу моделей может быть реализовано в рамках одной программной сборки .NET – Pa10.Modeling.dll (не считая нескольких необходимых ей сборок).

В какой-то степени, Pa10.Modeling можно считать аналогом открытого математического ядра ПА10. Используя ее, можно создать необходимую схему, запустить ее на расчет и получить все необходимые результаты без какого-либо взаимодействия с визуальной средой ПА10. Для этого нужно создать экземпляр компонента Схема (Pa10.Modeling.Schema), создать экземпляры необходимых функциональных элементов и объединить их должным образом. Затем следует создать экземпляр математического модуля (Pa10.Mathematics.DmanSolver), настроить его параметры и запустить схему на расчет.

#### Визуальное редактирование

#### Генерация кода

#### Загрузка и сохранение

#### Анализ данных

## Дизайн комплекса ПА10

### Дизайн-тайм компоненты ПА10

При работе с ПА10 можно условно выделить два режима взаимодействия с системой: режим разработки моделей и режим моделирования. В данной версии пользовательского интерфейса не проводится явного различения этих режимов, как это сделано, допустим, в пакете LabView. На текущем этапе схемы ПА10 не являются активными в том смысле, что пользователь не может взаимодействовать с моделируемой в реальном времени системой, переключая тумблеры или используя другие элементы управления.

Однако разделение на два таких режима должно присутствовать. Это объясняется тем, что работа системы по визуальному редактированию схемы и по обеспечению расчета параметров модели существенно различаются. Данный раздел вводит понятия, связанные с визуальным редактированием эквивалентных схем моделей в рамках программной модели ПА10, и содержит сведения, необходимые для создания специализированных компонентов и многополюсников.

#### Объекты режима редактирования

Редактирование элементов схемы обеспечивается большим числом сопутствующих объектов. В первую очередь, это различные дизайнеры. Поскольку данный вопрос требует более подробного изложения, лучше начать с примера.

Рассмотрим редактирование экземпляра Pa10.Modeling.Resistance, представляющего собой модель обобщенного резистора. Для обеспечения редактирования его в составе схемы, нужно реализовать следующую функциональность:

* Создание экземпляра;
* Отображение его на схеме в виде элемента управления, имеющего вид условного обозначения по ГОСТу;
* Изменение положения и ориентации условного изображения элемента на схеме;
* Редактирование свойств экземпляра резистора (имя на схеме и значение коэффициента сопротивления);
* Подключение к сопротивлению проводников.

Данная функциональность - общая для всех элементов схемы. Помимо нее, каждый элемент может обладать дополнительной, специфичной для него функциональностью. Для примера рассмотрим редактирование экземпляра хранилища решений, реализуемого классом Pa10.CalculationStorages.CalculationStorage. Хранилище решений сохраняет список нескольких предыдущих решений уравнений схемы. Для того чтобы иметь возможность сбора данных с индикаторов схемы, определять моменты старта и окончания процесса решения и т.д., хранилище решений должно взаимодействовать со страницей редактирования всей схемы.

Примеры с резистором и хранилищем решений наглядно показывают, что обеспечение редактирования является отдельной сложной задачей, не пересекающейся напрямую с математическими аспектами работы элементов эквивалентных схем. Для каждого элемента, помещенного на страницу редактирования, создаются как минимум два дополнительных объекта: экземпляр дизайнера модели (реализующего Pa10.Modeling.IModelDesigner) и экземпляр визуального дизайнера (реализующего Pa10.Modeling.IVisualDesigner). При этом сам экземпляр страницы редактирования экземпляра компонента доступен из обоих дизайнеров.

#### Дизайнер модели

Дизайнер модели обеспечивает правильное редактирование свойств экземпляра компонента и логическую связку его с другими объектами страницы редактирования. В частности, дизайнер модели определяет правила наименования экземпляров компонентов на странице.

#### Визуальный дизайнер

Визуальный дизайнер задает элемент управления, который отображает экземпляр компонента на странице редактирования. Управление изменением положения и ориентации, выделение и прочие визуальные аспекты в той или иной степени реализуются визуальным дизайнером.

### Страница редактирования

Экземпляр страницы редактирования в ПА10 является полным аналогом экземпляров класса Document для других систем из раздела «Дизайн современного ПО». Именно из него становятся доступными схема, все элементы схемы и сопутствующие им объекты. В таблице ниже приведен список свойств и методов страницы редактирования, с которыми будет непосредственно взаимодействовать любой программист серьезного нового компонента или расширения для ПА10.

ВСТАВИТЬ ТАБЛИЦУ СВОЙСТВ И МЕТОДОВ!!!

## Визуальный редактор функциональных эквивалентных схем (ФЭС)

Данный параграф посвящен вопросам проектирования и разработки схемного редактора ПА10 и некоторых других, связанных с ними, вопросов. В основе ПА10 являются новые математические методы моделирования, но основным инструментом работы инженера всё равно остается визуальная среда. Мощь и сложность математических методов, потенциал используемых моделей, должен быть наиболее полно и адекватно отображен визуальной средой, преобразован в многофункциональность и простоту и использования.

Схемный редактор ПА10 в первом приближении должен решать три задачи:

* визуальное редактирование свойств объектов, составляющих компонентную модель ПА10;
* визуальное редактирование содержимого страницы, под которым понимается изменение положения, размеров и других параметров визуальных элементов, отражающих любые редактируемые объекты;
* автоматическая прокладка проводников.

Рассмотрим ниже каждую из этих задач.

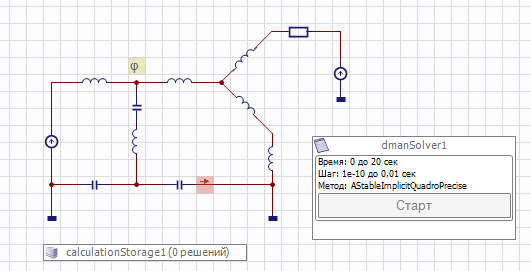
### Понятие страницы редактирования

При создании схемного редактора ПА10 использовалась одна важная абстракция, которая упрощает работу со схемой и пользователю, и программисту. Это абстракция *страницы редактирования*.

В самом общем смысле страница редактирования может быть представлена в виде листа бумаги, с которым привык работать инженер. На этом листе у него представлены и формулы, и графики, и небольшие чертежи. Так же там могут находиться списки неотложных дел, телефоны и e-mail адреса коллег, результаты промежуточных вычислений.

В этом аспекте ПА10 переняло лучшие традиции своего предшественника – комплекса ПА9.

Любой объект может быть помещен на станицу. В итоге работа с ПА10 заключается в создании пустой страницы и накидывании на него множества различных объектов, от элемента функциональной схемы, до графиков и Интернет адресов.



1. Пример внешнего вида страницы в ПА10

Как и в ПА9, компонент, обеспечивающий вычислительные способности комплекса ПА10, является редактируемым объектом страницы, что позволяет оперировать сразу несколькими настройками параметров интегрирования. Такая функциональность часто бывает просто необходима и избавляет инженера от постоянного переключения между настройками системы, например, в главном меню приложения.

### Редактирование схем в ПА10

Редактирование эквивалентных функциональных схем в ПА10 сводится к выполнению определенного набора операций. Задавшись целью провести анализ некоторой модели, инженер должен:

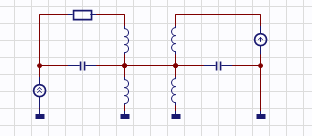
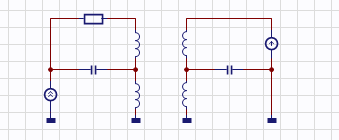
* Поместить на страницу редактирования необходимые элементы из браузера компонентов;
* Соединить элементы в схему при помощи проводников;
* Настроить свойства элементов, задав, таким образом, параметры модели;
* Поместить на страницу редактирования необходимые индикаторы и настроить их свойства;
  + Индикаторы прикрепляются к узлам и контактам схемы простым перемещением их в область нужного проводника;
  + Изображение индикатора, подключенного к узлу или контакту, имеет закрашенное внутреннее пространство. Неподключенный индикатор отображает только рамку.
* Поместить на страницу редактирования блок из *математического компонента*, *хранилища решений* и *графика*. Настроить параметры расчета и запустить процесс решения, нажав «Вычислить» на элементе управления экземпляром математического компонента.

Рассматривая особенности работы со схемами, стоит отметить несколько важных вопросов, непосредственно касающихся схем и возможности их решения. Работая с ПА10 пользователь ожидает, что система автоматически отбросит «мусор» в виде отдельно расположенных и не подключенных ни к одной цепи компонентов. Для обеспечения этой функциональности, должен быть написан специальный программный код проверки и сборки связных схем.

Помимо вопроса отношений с пользователем, есть еще один аспект, напрямую вытекающий из компонентной архитектуры ПА10. Любой квалифицированный человек может создать для ПА10 совершенно новый компонент. Соответственно, разработчики комплекса не могут полагаться на свое полное знание всех типов элементов, и не могут написать не полиморфный код, который бы собирал связную схему из неизвестных ему компонентов. Решение о том, какой элемент войдет в схему, а какой оказался неподключенным к ней, наиболее просто может быть решен не в процессе редактирования схемы, а в начала процесса расчета. На практике это вносит элемент потери производительности системы при каждом старте вычисления.

Этот вопрос требует более глубокого исследования. Известно, что множество всех возможных графических представлений эквивалентных схем и множество всех вычислимых эквивалентных схем не совпадают. Т.е. всегда можно изобразить схему, не имеющую физического смысла или неподвластную существующей математической модели. Поэтому часто возникают ситуации, когда незначительное изменение графического представления значительно меняет представляемую ею математическую модель.

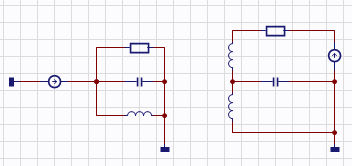
Самым простым примером может послужить соединение двух независимых цепей проводником. При этом из двух существовавших узлов остается только один.



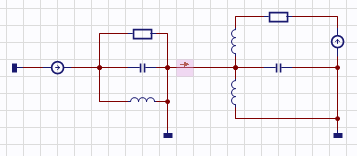
1. Две цепи до и после соединения

Визуально на схеме остаются два узла, однако нижележащая модель содержит только один узел. Адекватно разобраться с этой ситуацией проще всего на этапе анализа схемы перед началом вычисления, при котором просто создается новая схема, удовлетворяющая требованиям модели.

Еще более интересные ситуации могут возникнуть с индикаторами потока, прицепленными к таким соединяющим проводникам.



1. Схема до соединения цепей



1. Схема после соединения цепей и помещения индикатора

В действительности, индикатор на рисунке снимает значение потока, исходящего их правой ножки левого конденсатора, а не поток, переходящий из одной схемы в другую, как может показаться. Принципиально можно добиться того, чтобы индикатор показывал поток между цепями, но практически это потребует отдельного исследования «правильного» поведения системы во множестве схем. Вероятно, более правильным будет активное использование понятных обозначений и готового шарокого набора многополюсников, не допускающих ошибок в интерпретации.

### Работа с компонентами

В данном разделе под компонентами и их экземплярами понимаются функциональные элементы, которые используются инженером при построении и анализе моделей. В частности, это элементы функциональных схем, проектируемые многополюсники, полученные решения, графики и т.д. Объектно-ориентированный подход в применении к таким компонентам заключается в том, чтобы предоставить возможность использования каждого их них в качестве самостоятельной единицы проекта.

Экземпляр диода, включенного в проект наряду со схемами моделей, задает конкретную модель реально существующего диода путем фиксации значений его параметров. Класс, автоматически создаваемый под данный диод, будет являться наследником обобщенной модели диода, и будет фиксировать значения свойств с ограничением возможности их изменения. Также могут быть заданы отклонения от эталонных значений для промышленных образцов. Использование экземпляров нового класса в дальнейшем при построении схем будет ограниченным в том смысле, что они смогут изменять значения своих свойств только в пределах заданных отклонений.

Экземпляр графика, включенного в схему, будет задавать настройки отображения графиков – цвета фона и рамки, толщину линий и т.д. Экземпляры класса, сгенерированного по данному графику, могут быть помещены на схему и будут при этом сразу выглядеть надлежащим образом, а возможно и отображать какую-то эталонную кривую.

### Работа с решениями

В большинстве существующих электрических CAE решение уравнений схемы отображается в окне с графиком, появляющимся вслед за нажатием кнопки «Run», «Start» или «Dynamic». При этом схема оказывается загороженной модальным окном решения и, что важно, старое решение пропадает (если не было отдельно сохранено в файл). Сравнить предыдущее решение схемы с текущим решением после ее модификации становится непросто. В целом, работа с решениями слабо представлена как в PCAD, PSPICE, так и в ПА9.

В дизайнере схем ПА10 предлагается использовать явный подход к работе с решениями. Под решением понимается набор данных, полученных в результате решения системы уравнений схемы, объединенный с дополнительной информацией об этих данных, такой как:

* Наименование значений и единицы измерения;
* Время получения решения;
* Наименование схемы, для которой было получено данное решение.

Последовательно продолжая линию «все есть компонент», решение в ПА10 также реализовано в виде компонента. Для работы с решениями вводятся несколько дополнительных компонентов.

#### Хранилище решений

Внешний вид экземпляра хранилища решений представлен на рисунке ниже.



1. Внешний вид экземпляра хранилища решений

Будучи помещенным на страницу редактирования, хранилище решений начинает собирать данные с индикаторов схемы. Для каждого цикла моделирования, хранилище решения собирает временной ряд значений индикаторов, отмеченных в данном хранилище как подлежащие мониторингу. Собрав временной ряд значений, хранилище создает экземпляр решения и сохраняет его в себе.

В качестве одного из свойств хранилища выступает максимальное количество последних решений, которые необходимо сохранять. Т.е. когда в новое решение поступает в наполненное хранилище, наиболее старое решение удаляется. Теперь решения из хранилища можно поместить на график, сохранить в файл или сделать частью проекта.

#### Браузер решений

Браузер решений представляет собой аналог хранилища решений для всех схем проекта сразу. Браузер отображает последние решения всех схем и дополнительные решения, если их визуально поместить в него из любого хранилища решений (или загрузить с диска).

#### Отображение графиков

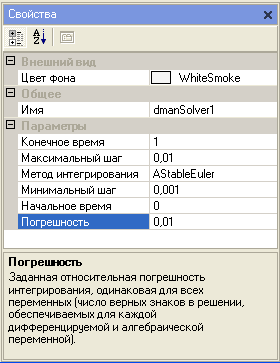
Полученные решения отображаются в компоненте «график». Экземпляры данного компонента могут быть помещены на страницу редактирования в любом количестве, и будут отображать графики помещенных в них решений. Также можно выбрать режим, в котором компонент отображает график последнего (по времени) решения, сохраненного в некотором хранилище схемы.

В результате, инженеру предоставляется возможность работы с несколькими решениями в рамках одного компонента отображения графиков, сравнить решения для разных значений параметра элемента схемы или с некоторой эталонной кривой, полученной в другой схеме или даже другой CAE.

Во многом приведенный подход схож с работой в пакете MathCAD фирмы M.

### Редакторы свойств

При помещении объекта на страницу, для него создаются два дополнительных программных объекта-редактора: *визуальный редактор* и *редактор модели*. Визуальный редактор создает для редактируемого объекта визуальный компонент и помещает его на страницу, чтобы пользователь мог ощущать редактируемый объект как отдельную сущность, часть листа страницы. Редактор модели отвечает за состав свойств объекта, доступных для редактирования в *редакторе свойств*.



1. Редактор свойств в ПА10

Редактор свойств, как он виден пользователю, представляет собой таблицу из двух колонок, отображающих названия и значения всех доступных свойств компонента. Данный редактор полностью аналогичен тому, что используется в продукте Microsoft© Visual Studio© 2003.

Таким образом, пользователю в каждый момент времени непосредственно доступны все объекты и абстракции, с которыми он работает при проведении моделирования.

Для разработчика компонентов ПА10 такой подход означает, что ему достаточно разработать два класса объектов-редакторов, после чего новый компонент может быть безболезненно внедрен в среду и сможет полноценно работать в окружении уже существующих компонентов на равных правах.

Как видно из рисунка, редактор свойств предоставляет расширенную информацию о каждом из них, обеспечивая пользователю комфортную работу в незнакомой среде. Настраивая свойства редактируемых компонентов, можно легко связать их в необходимую структуру.

### Трассировка проводников

Проводник в ПА10, как и страница редактирования, представляет некоторую общую абстракцию семантического связывания объектов страницы. Проводник может быть направленным и передавать поток данных. Может связывать объекты в группу. И, конечно, проводник может представлять собой модель идеального проводника при соединении элементов ФЭС.

Соответственно трассировка проводника в ПА10 представляет собой общую задачу прокладки на плоскости строго направленного пути через множество препятствий.

#### Общие сведения

Трассировка (или прокладка) проводников является давно известной и хорошо изученной задачей. К сожалению, гарантированная оптимальная трассировка всех проводников схемы до сих пор, в той или иной степени, требует полного перебора вариантов. На данном этапе развития вычислительной техники, полный перебор недопустим.

Подавляющее большинство подсистем трассировки современных CAE используют модификации идеи Ли о распространении фронта абстрактной числовой волны – *волновые алгоритмы*.

В общем случае, алгоритм работает в два шага. На первом шаге происходит распространение волны, т.е. поиск соединяемыми элементами друг друга. На втором этапе происходит трассировка провода в соответствии с некоторым градиентом от построенной волны на плоскости. Если первый этап можно сравнить с растеканием жидкого воска, то второй схож с последующим растеканием воды ручьями по углублениям застывшего воска.

Волновые алгоритмы допускают оптимизацию трассируемого пути. В грубом приближении, все используемые критерии оптимизации можно разбить на две группы: сосредоточенные и интегральные. Сосредоточенные критерии применимы непосредственно к каждой точке плоскости. Интегральные критерии применимы только к какому-то из возможных путей в целом. Например: предпочтительная ориентация пути – это сосредоточенный критерий в каждой точке, а минимальная длина провода – интегральный.

Некоторой разновидностью волнового алгоритма можно считать *лучевой алгоритм* трассировки. Данный алгоритм связан с распространением лучей по приоритетным направлениям и требует значительно меньше памяти, чем волновой.

Также большое распространение получили генетические алгоритмы применительно к задаче канальной трассировки.

#### Требования к процессу трассировки

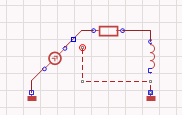
Применительно к ПА10 задача трассировки должна обеспечивать следующие возможности:

1. Быструю интерактивную прокладку проводника;
2. Возможность указать предпочтительный путь трассировки;
3. Возможность редактировать проложенный путь и не терять этой информации при выполнении повторной трассировки;
4. Учитывать ориентацию элементов. Т.е. угол подвода провода к контакту может быть произвольным.

Проще говоря, трассировка должна максимально облегчать работу с системой, не сковывать возможностей пользователя и не раздражать его необходимостью постоянно править проложенный путь.

#### Реализация алгоритма трассировки

В данной редакции ПА10 принят не самый оптимальный *лучевой алгоритм*, удовлетворяющий только последним трем пунктам. В ситуации наличия двух-трех видимых препятствий трассировка начинает заметно снижать работоспособность всей системы. Для того чтобы не нарушать требования интерактивности, в систему введено ограничение на полное время трассировки. Таким образом, даже в самом худшем случае время трассировки не превышает трех (3) сек.



1. Пример соединения контактов

#### Дальнейшее развитие

В качестве основной задачи развития существующей системы можно назвать задачу существенного ускорения процесса трассировки. Требование интерактивности можно будет считать удовлетворенным при десятикратном ускорении.

## Сравнение возможностей систем серии ПА

Главным и основным достоинством комплекса ПА10 является возможность визуального проектирования схем и *компонентов* функциональных эквивалентных схем.

1. Сравнение возможностей систем серии ПА

|  |  |  |  |
| --- | --- | --- | --- |
|  | ПА7 | ПА9 | ПА10 v0.1 |
| **Моделирование** |  |  |  |
| Моделирование гетерогенных систем | + | + | + |
| Поддержка единиц измерения | + | + | + |
| Проверка целостности модели | + | + | + |
| **Вычислительные возможности** |  |  |  |
| Адаптивные методы решения систем | - | - | + |
| Оптимизация под конкретный процессор | - | - | + |
| **Редактирование моделей** |  |  |  |
| Встроенный язык описания моделей | + | + | - |
| Визуальное редактирование моделей | - | - | + |
| Работа с графиками | - | + | + |
| Автоматическая трассировка проводников | - | - | + |
| **Разработка компонентов** |  |  |  |
| Разработка новых компонентов | + | + | + |
| Язык разработки компонентов | Си-подобный | Java-подобный | Любой CLR-совместимый язык (C#, VB.NET, C++, J#, FORTRAN и т.д.) |
| Визуальная разработка компонентов | - | - | + |
| **Переносимость** |  |  |  |
| Работа под DOS | + | - | - |
| Работа под Win32 | + | + | + |
| Работа под Linux | - | + | + |
|  |  |  |  |

# Приложения

## Приложение 1. Google Citation Index

28 апреля 2005

Результаты **1** - **10** из примерно **2 915 000** для **AutoCAD**. (**0,23** секунд)

Результаты **1** - **10** из примерно **520 000** для **Microstation**. (**0,24** секунд)

Результаты **1** - **10** из примерно **460 000** для **SolidWorks**. (**0,32** секунд)

Результаты **1** - **10** из примерно **412 000** для **Unigraphics**. (**0,38** секунд)

Результаты **1** - **10** из примерно **223 000** для **PSPICE**. (**0,35** секунд)

Результаты **1** - **10** из примерно **109 000** для **SolidEdge**. (**0,36** секунд)

Результаты **1** - **10** из примерно **74 200** для **PCAD**. (**0,34** секунд)

Результаты **1** - **10** из примерно **141** для **ПА9**. (**0,53** секунд)

Результаты **1** - **10** из примерно **130** для **ПА7**. (**0,29** секунд)

24 мая 2005

Результаты **1** - **10** из примерно **3 160 000** для **AutoCAD**. (**0,23** секунд)

Результаты **1** - **10** из примерно **824 000** для **Microstation**. (**0,24** секунд)

Результаты **1** - **10** из примерно **751 000** для **SolidWorks**. (**0,32** секунд)

Результаты **1** - **10** из примерно **458 000** для **Unigraphics**. (**0,38** секунд)

Результаты **1** - **10** из примерно **215 000** для **PSPICE**. (**0,35** секунд)

Результаты **1** - **10** из примерно **146 000** для **SolidEdge**. (**0,36** секунд)

Результаты **1** - **10** из примерно **77 800** для **PCAD**. (**0,34** секунд)

Результаты **1** - **10** из примерно **202** для **ПА9**. (**0,53** секунд)

Результаты **1** - **10** из примерно **129** для **ПА7**. (**0,29** секунд)

Примерно **545 000** для **AutoCAD addin OR addon OR add-in OR add-on OR Plugin OR Plug-in**.

Примерно **44 200** для **Microstation addin OR addon OR add-in OR add-on OR Plugin OR Plug-in**.

Примерно **96 100** для **SolidWorks addin OR addon OR add-in OR add-on OR Plugin OR Plug-in**.

Примерно **26 400** для **Unigraphics addin OR addon OR add-in OR add-on OR Plugin OR Plug-in**.

Примерно **11 100** для **PSPICE addin OR addon OR add-in OR add-on OR Plugin OR Plug-in**.

Примерно **20 000** для **SolidEdge addin OR addon OR add-in OR add-on OR Plugin OR Plug-in**.

Примерно **978** для **PCAD addin OR addon OR add-in OR add-on OR Plugin OR Plug-in**.

28 апреля 2005

Результаты **1** - **10** из примерно **448 000** для **AutoCAD addin OR addon OR add-in OR add-on OR Plugin OR Plug-in**. (**0,98** секунд)

**1:** [Autodesk - **AutoCAD** - Viewers](http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=2952277&linkID=2475161)  
Download the Autodesk DWF Viewer - intelligence you can see!  
usa.autodesk.com/adsk/servlet/ index?siteID=123112&id=2952277&linkID=2475161

**542:** [AcadExcel](http://www.shareup.net/Multimedia/Image-Editing/AcadExcel-review-18360.html)  
AcadExcel - AcadExcel is an **AutoCAD** **Add-In**, designed for process table in **AutoCAD**.  
Using our product, you can use MicroSoft Excel as a table editor.  
www.shareup.net/Multimedia/ Image-Editing/AcadExcel-review-18360.html

Результаты **1** - **10** из примерно **40 000** для **Microstation addin OR addon OR add-in OR add-on OR Plugin OR Plug-in**. (**0,93** секунд)

**1:** [Surpac Software International On-line Help](http://www.surpac.com/refman/default/data_manager/plugins/microstation.htm)  
**...** the method used for reading and saving files using the **Microstation** **plugin**.  
Reading. All data in a **Microstation** design file is read in as String data. **...**  
www.surpac.com/refman/default/ data\_manager/**plugin**s/**microstation**.htm

**726:** [**MicroStation** Computers, Directory](http://www.sigmbi.org/c2lnXzYwNTI%3D.aspx)  
**...** **Microstation** **add-on** software packages for piping and structural steel. **...**  
Ltd. Specialises in customizing **MicroStation**, with database applications **...**  
[www.sigmbi.org/c2lnXzYwNTI%3D.aspx](http://www.sigmbi.org/c2lnXzYwNTI%3D.aspx)

Результаты **1** - **10** из примерно **79 600** для **SolidWorks addin OR addon OR add-in OR add-on OR Plugin OR Plug-in**. (**0,85** секунд)

**1:** [**SolidWorks**/GibbsCAM Interoperability Streamlined with Free **Add-In** **...**](http://www.gibbscam.com/gibbs/press/press100501.asp)  
**...** announced today that they are now offering an **add-in** for **SolidWorks**® **...**  
The GibbsCAM **SolidWorks** **Add-In** facilitates one-button transfer of model files **...**  
[www.gibbscam.com/gibbs/press/press100501.asp](http://www.gibbscam.com/gibbs/press/press100501.asp)

**499:** [Hewlett Packard Graphics Language Software at FilesLand](http://www.filesland.com/software/hewlett-packard-graphics-language.html)  
**...** HPGL Import for **SolidWorks** is an **add-in** for **SolidWorks**. **...** open file plt plot  
file rhino plug-in **plugin** **add-in** **addin** hewlett packard graphics language **...**  
www.filesland.com/software/ hewlett-packard-graphics-language.html

Результаты **1** - **10** из примерно **52 400** для **Unigraphics addin OR addon OR add-in OR add-on OR Plugin OR Plug-in**. (**1,29** секунд)

**1:** [eDrawings **add-in** for SolidWorks, AutoCAD, CATIA, Pro/ENGINEER **...**](http://www.edrawingsviewer.com/pages/faqs/)  
**...** AutoCAD®, Invetor®, Pro/ENGINEER® CATIA® **Unigraphics**/NX®,Cocreate - once space  
**...** Publisher **add-in**, which works as a plug-in to popular CAD products. **...**  
[www.edrawingsviewer.com/pages/faqs/](http://www.edrawingsviewer.com/pages/faqs/)

**695:** [IT BASE "sense & logic" Avec NORMICA VIEW , visualisez les **...**](http://www.normica.net/doc_fr/30_fr_nview.htm)  
**...** manuals,SCP,Plancom,coala,macron,AutoCAD,Microstation,**Unigraphics**,CAD400,Nemetschek  
**...** (\*Netscape ou Internet Explorer ainsi que **plug-in** Whip gratuit). **...**  
[www.normica.net/doc\_fr/30\_fr\_nview.htm](http://www.normica.net/doc_fr/30_fr_nview.htm)

Результаты **1** - **10** из примерно **5 060** для **PSPICE addin OR addon OR add-in OR add-on OR Plugin**. (**0,42** секунд)

**1: [PDF]** [DEIC420 Spice Model](http://www.ixysrf.com/pdf/switch_mode/appnotes/9ap_deic420_spice_model.pdf)  
Формат файла: PDF/Adobe Acrobat - [В виде HTML](http://216.239.59.104/search?q=cache:zKsArRTAeAUJ:www.ixysrf.com/pdf/switch_mode/appnotes/9ap_deic420_spice_model.pdf+PSPICE+addin+OR+addon+OR+add-in+OR+add-on+OR+Plugin&hl=ru)  
**...** The following application note describes a **PSpice** model for the DEIC420 and  
**...** This model was developed using **Pspice** **Plugin** version. 9.2.3.268. **...**  
www.ixysrf.com/pdf/switch\_mode/ appnotes/9ap\_deic420\_spice\_model.pdf

**663:** [Programme - Brennen, Rippen, Streming, Filesharing,...](http://progs.labz.net/)  
**...** **Plugin** Problem (plugger 4.0) 4865. Joe ausgabe kopieren und einfügen 4866.  
**...** **PSpice** 12572. Software für Kleinunternehmen 12573. **...**  
progs.labz.net/ - 101k - 26 АПР 2005

Результаты **1** - **10** из примерно **9 370** для **SolidEdge addin OR addon OR add-in OR add-on OR Plugin**. (**0,56** секунд)

**1:** [CATIA2SolidEdge Bi-directional](http://www.radialsoft.com/products/catia2se_bidirect.htm)  
**...** and automatically translates **SolidEdge** parts and assemblies into CATIA models.  
As a **SolidEdge** **add-in**, CATIA2SolidEdge is fully integrated with **SolidEdge** **...**  
www.radialsoft.com/products/catia2se\_bidirect.htm

**484:** [SE BOM Extractor - Download SE BOM Extractor 3.0.15 Free Trial](http://www.softandco.com/a/20143/se-bom-extractor.html)  
**...** **SolidEdge** Automation Tools that easily extracts Bill of Materials from **SolidEdge**  
files for easy integration with 3rd party software, like ERP and SCM **...**  
[www.softandco.com/a/20143/se-bom-extractor.html](http://www.softandco.com/a/20143/se-bom-extractor.html)

Результаты **1** - **10** из примерно **545** для **PCAD addin OR addon OR add-in OR add-on OR Plugin**. (**0,34** секунд)

**1:** [Caxapa - Схемы - P-CAD Desktop (for **PCAD** rev 6.0-8.7)](http://www.caxapa.ru/sch/pcad.html)  
**...** Support **PCAD** rev 6.0-8.7 - View and print in color with WINDOWS fonts and **...**  
Links (\*.lnk \*.pif \*.url) from subfolder \Add-ins **add** **in** program menu **...**  
[www.caxapa.ru/sch/**pcad**.html](http://www.caxapa.ru/sch/pcad.html)

**373:** [::mp3000.net:: free mp3 downloads, latest mp3 songs, full mp3 **...**](http://www.mp3000.net/)  
MP3 search engine with direct downloads. Includes a browsable music archive and  
a lyrics database.  
[www.mp3000.net/](http://www.mp3000.net/)

Результаты **1** - **10** из примерно **408 000** для **AutoCAD library -book**. (**0,28** секунд)

**1:** [**AutoCAD** Detail and Symbol **Library**](http://www.autocadlibrary.com/)  
Over 4000 high quality drawing symbols, details and blocks on CD, which can be  
modified for you own application. All the architectural, interior and **...**  
[www.**autocadlibrary**.com/](http://www.autocadlibrary.com/)

**742:** [Illustration Software Download](http://www.shareup.net/Multimedia/Illustration/index-103-1.html)  
**...** DSYMS for **AutoCAD** LT is a single Pulldown Menu symbols **library** Add-on for  
**AutoCAD** LT2004, geared towards the 2D ARCHITECTURAL and Interiors disciplines. **...**  
www.shareup.net/Multimedia/ Illustration/index

Результаты **1** - **10** из примерно **46 400** для **Microstation library -book**. (**0,27** секунд)

**1:** [NHDOT File **Library**](http://webster.state.nh.us/dot/its/cadd/cadd.html)  
Welcome to the New Hampshire Department of Transportation File **Library** **...**  
Word documents read in WordPerfect, **MicroStation** design files to AutoCAD). **...**  
webster.state.nh.us/dot/its/cadd/cadd.html - 26 АПР 2005

**754:** [Architectural **Microstation** Technician jobs Cambridgeshire jobs **...**](http://ukworksearch.co.uk/Construction/Cambridgeshire/Job/36202/)  
**...** Must be proficient on **Microstation** and ideally have experience of education  
**...** CV-**Library**.co.uk is one of the leading online recruitment job banks, **...**  
ukworksearch.co.uk/Construction/ Cambridgeshire/Job/36202/

Результаты **1** - **10** из примерно **63 300** для **SolidWorks library -book**. (**0,29** секунд)

**1:** [**SolidWorks** COSMOSWorks Data Export](http://www.matweb.com/help/SolidWorks_intro.asp)  
**...** The next step to the export process is to create a new **SolidWorks** **library**.  
**...** Save the **SolidWorks** **Library** additions by selecting "Save SW **Library**" from **...**  
[www.matweb.com/help/SolidWorks\_intro.asp](http://www.matweb.com/help/SolidWorks_intro.asp)

**691: [PDF]**[Supply Chain Link June'03](http://www2.anisagroup.com/newsletter/anisa_group_newsletter_issue_7.pdf)  
Формат файла: PDF/Adobe Acrobat - [В виде HTML](http://216.239.59.104/search?q=cache:UXHxMKoqZdYJ:www2.anisagroup.com/newsletter/anisa_group_newsletter_issue_7.pdf+SolidWorks+library+-book&hl=ru)  
**...** Ireland’s North Eastern Education and **Library** **...** Northern Ireland’s Classroom  
2000 initiative standardises on **SolidWorks** **...**  
www2.anisagroup.com/newsletter/ anisa\_group\_newsletter\_issue\_7.pdf

Результаты **1** - **10** из примерно **22 300** для **Unigraphics library -book**. (**0,27** секунд)

**1:** [UGS: Produkte & Lösungent: NX: **Unigraphics** NX: Digital Manufacturing](http://www.ugsplm.de/produkte/nx/unigraphics/dm/advanced_kinematics.shtml)  
**...** Produkte & Lösungen:; NX:; **Unigraphics** NX:; Digital Manufacturing - Advanced  
Kinematics **Library**. nx logo. NX - Digital Manufacturing **...**  
www.ugsplm.de/produkte/nx/ **unigraphics**/dm/advanced\_kinematics.shtml - 22k - 26 АПР 2005

**716:** [Open Directory Project > Computers> CAD and CAM> **Unigraphics** and **...**](http://www.bie.no/products/phpodp/odp.php?browse=/Computers/CAD_and_CAM/Unigraphics_and_Solid_Edge/)  
**...** **UNIGRAPHICS** TO SUPPLY MECHANICAL CAD/80 SEATS OF SOLID EDGE. **...** parts and  
**library** management, printing, performance optimization, and related tasks. **...**  
www.bie.no/products/phpodp/odp.php?browse=/ Computers/CAD\_and\_CAM/Unigraphics\_and\_Solid\_Edge/

Результаты **1** - **10** из примерно **19 000** для **PSPICE library -book**. (**0,42** секунд)

**1: [PDF]** [TANTALUM AND NIOBIUM OXIDE DATA LIBRARIES FOR INSERTION INTO **...**](http://www.avxcorp.com/SpiApps/pspice/pspintro.pdf)

Формат файла: PDF/Adobe Acrobat - [В виде HTML](http://216.239.59.104/search?q=cache:3G6jm4bjfw8J:www.avxcorp.com/SpiApps/pspice/pspintro.pdf+PSPICE+library+-book&hl=ru)  
**...** SLB **library** files into the **PSpice** **library** directory. **...** A **PSpice** **library** of electronic components is a useful tool for fast, flexible electronic **...**  
www.avxcorp.com/SpiApps/**pspice**/pspintro.pdf

**799:** [Ansoft Coperation](http://www.ansoft.co.kr/product/spicelink.html)

**...** User Defined Primitives (UDP) for customizable elements/**library** of parts **...** 또한 이들 등가회로는 SPICE 호환이 되므로 HSpice, **PSpice** 혹은 다른 **...**  
www.ansoft.co.kr/product/spicelink.html

Результаты **1** - **10** из примерно **4 760** для **SolidEdge library -book**. (**0,29** секунд)

**1:** [CADalog Inc. - Part **Library** Development Specialist](http://www.cadalog-inc.com/software.html)  
**...** 3D parts libraries for SolidWorks, AutoCAD, Inventor and **SolidEdge**. Originally  
developed, the parts **library** applications produce native, feature-based, **...**  
www.cadalog-inc.com/software.html

**605:** [Development Computers, Directory](http://www.sigmbi.org/c2lnXzEwMTE2OQ%3D%3D.aspx)  
**...** KolbaSoft The VeCAD 2D vector graphics **library** can be used to create CAD **...**  
Parasolid The geometry engine inside SolidWorks, **SolidEdge**, DesignWave, **...**  
[www.sigmbi.org/c2lnXzEwMTE2OQ%3D%3D.aspx](http://www.sigmbi.org/c2lnXzEwMTE2OQ%3D%3D.aspx)

Результаты **1** - **10** из примерно **3 560** для **PCAD library -book**. (**0,72** секунд)

**1:** [Создание библиотечного компонента в P-CAD 2001 **Library** Executive.](http://www.chipinfo.ru/literature/chipnews/200205/14.html)  
**...** Но после обработки компонента инструментами **PCAD** **Library** Executive эти пробелы  
**...** компонентов командой Delete из меню **Library** в программе **PCAD** **Library** **...**  
www.chipinfo.ru/literature/chipnews/200205/14.html

**634:** [Science Social Sciences Methodology Qualitative Tools - infopool-24.de](http://www.infopool-24.de/index.php/Science/Social_Sciences/Methodology/Qualitative/Tools)  
**...** http://www.gb-software.com/, » **PCAD** 2000 In neuem Fenster öffnen **...**  
Top/Reference/Libraries/**Library** and Information Science/Software **...**  
www.infopool-24.de/index.php/Science/ Social\_Sciences/Methodology/Qualitative/Tools

## Приложение 2. Пример кода сборки и решения эквивалентной схемы на C#.

 // создание элементов.  
 BaseNode b1 = new BaseNode();  
 BaseNode b2 = new BaseNode();  
 BaseNode b3 = new BaseNode();

 Node n1 = new Node();  
 Node n2 = new Node();  
 Node n3 = new Node();  
 Node n4 = new Node();  
 Node n5 = new Node();  
 Node n6 = new Node();  
  
 USource uSource = new USource(1);

 Resistance r1 = new Resistance(1);  
 Resistance r2 = new Resistance(1);  
  
 Inductance i1 = new Inductance(1001);  
 Inductance i2 = new Inductance(999);  
  
 Condenser c1= new Condenser(0.001);  
 Condenser c2= new Condenser(1);  
 Condenser c3= new Condenser(0.001);  
  
 // соединение элементов с узлами  
 b1.ConnectedContacts.AddRange(uSource.InContact, c2.InContact, r2.OutContact);  
 n1.ConnectedContacts.AddRange(uSource.OutContact, r1.InContact);  
 n2.ConnectedContacts.AddRange(r1.OutContact, i1.InContact);  
 n3.ConnectedContacts.AddRange (i1.OutContact, c1.InContact);  
 n4.ConnectedContacts.AddRange (c1.OutContact, c2.OutContact, c3.InContact);  
 n5.ConnectedContacts.AddRange (c3.OutContact, i2.InContact);  
 n6.ConnectedContacts.AddRange (i2.OutContact, r2.InContact);

// подключение индикаторов  
 PotentialIndicator in3 = new PotentialIndicator(n3);  
 PotentialIndicator in4 = new PotentialIndicator(n4);  
 PotentialIndicator in5 = new PotentialIndicator(n5);  
     
 // Решение:  
 // создание экземпляра вычислительного компонента и настройка параметров расчета.  
 Pa10.Mathematics.DmanSolver solver = new Pa10.Mathematics.DmanSolver();     
 solver.Schema = Schema.Collect(b1);

 solver.DeltaTimeMin = 0.001;  
 solver.Epsilon =0.01;  
 solver.EndTime = 128;  
 solver.DeltaTimeMax = 128.0D;  
 solver.DeltaTimeMin = 128.0D/1e12;  
 solver.IntegrationMethod = Pa10.Mathematics.IntegrationMethod.AStableImplicitQuadroPrecise;  
  
 // запуск на расчет  
 solver.Evaluate();  
  
 // сбор результатов  
 double[] res3 = in3.GetMeasurings();  
 double[] res4 = in4.GetMeasurings();  
 double[] res5 = in5.GetMeasurings();

## Приложение 3. Пример сборки схемы на XML

<?xml version="1.0" encoding="utf-8" ?>

<schema xmlns="http://www.pa10.org/xsdschemas/Schema.xsd">

<elements>

<Usource id="uSource" E="1" />

<Resistance id="r1" R="1" />

<Resistance id="r2" R="1" />

<Inductance id="i1" L="1001" />

<Inductance id="i2" L="999" />

<Condenser id="c1" C="0.001" />

<Condenser id="c2" C="1" />

<Condenser id="c3" C="0.001" />

<BaseNode id="b1">

<connect element="uSource" contact="InContact" />

</BaseNode>

<BaseNode id="b2">

<connect element="c2" contact="InContact" />

</BaseNode>

<BaseNode id="b3">

<connect element="r2" contact="OutContact" />

</BaseNode>

<Node id="n1">

<connect element="uSource" contact="OutContact" />

<connect element="r1" contact="InContact" />

</Node>

<Node id="n2">

<connect element="r1" contact="OutContact" />

<connect element="i1" contact="InContact" />

</Node>

<Node id="n3">

<connect element="i1" contact="OutContact" />

<connect element="c1" contact="InContact" />

</Node>

<Node id="n4">

<connect element="c1" contact="OutContact" />

<connect element="c2" contact="OutContact" />

<connect element="c3" contact="InContact" />

</Node>

<Node id="n5">

<connect element="c3" contact="OutContact" />

<connect element="i2" contact="InContact" />

</Node>

<Node id="n6">

<connect element="i2" contact="OutContact" />

<connect element="r2" contact="InContact" />

</Node>

<PotentialIndicator id="in3" node="n3" />

<PotentialIndicator id="in4" node="n4" />

<PotentialIndicator id="in5" node="n5" />

</elements>

<solution>

<DmanSolver id="solver" Epsilon="0.01" EndTime="128" DeltaTimeMax="128.0"

DeltaTimeMin="128.0e-12" IntegrationMethod="AStableImplicitQuadroPrecise" />

</solution>

</schema>

1. Архитектура (eng.) – архитектура программного обеспечения. [↑](#footnote-ref-1)
2. Строить (eng.) – процесс компиляции и сборки проекта, результат этого процесса. [↑](#footnote-ref-2)
3. Каркас, стропила (eng.) – библиотека классов, предназначенная для разработки определенного типа программ. [↑](#footnote-ref-3)
4. т.к. на диаграммах нельзя толком ничего протестировать, а если можно, то это уже не диаграмма, а полноценная программа. Попытки заставить диаграммы работать как полноценные программы и автоматически генерировать программный код начинаются во многих компаниях. Компания Rational Software обещает, что с внедрением их продуктов доля автоматически генерируемого кода достигает 80%. [↑](#footnote-ref-4)
5. Строительство «Титаника» потребовало примерно 10 млн. болтов и заклепок, согласно книге Бережных О.А. «Самые большие корабли», изд. Судостроение, Ленинград, 1985. [↑](#footnote-ref-5)
6. См. Проекты Pink от Apple и BeOS от BeOS Software, Джим Карлтон, «Apple. Взгляд изнутри: история интриг, ошибок и эгоизма», Лори, 2001. А также проект PC от Hulled & Packard: Пол Фрейбергер и Майкл Свейн, «Пожар в долине», группа Дарнел, 2000. [↑](#footnote-ref-6)
7. См. успех подхода RUP, Грэди Буча и фирмы Rational Software. [↑](#footnote-ref-7)
8. Данные по исследованию «Extreme Chaos», Standish Group International, Inc. (коммерческая организация, занимающаяся консультированием в области инвестиций в информационные технологии), 2001. [↑](#footnote-ref-8)
9. В простом изложении антропный принцип Брэндона Картера звучит примерно так: «Вопрос о том, почему наш мир и планета так хорошо устроены под людей, возникает только потому, что в таких условиях появился тот, кто может его задать. В других, плохих, условиях, некому было бы задаться этим вопросом». [↑](#footnote-ref-9)
10. Генотип - наследственная информация, закодированная в генах. [↑](#footnote-ref-10)
11. Фенотип - совокупность характеристик, присущих индивиду на определённой стадии развития. [↑](#footnote-ref-11)
12. Т.е. старая и новая версии генетического или программного кода. [↑](#footnote-ref-12)
13. Burroughs 220 – ламповый компьютер фирмы Burroughs Corporation, расположенной в Детройте. Разработан в 1957 году компанией Electrodata Corporation и назывался Electrodata Datatron 220 до того, как Electrodata была поглощена Burroughs. В разное время стоил от $601 000 до $325 000 и выполнял умножение за 2, а деление почти за 4 микросекунды. [↑](#footnote-ref-13)
14. Подробнее об объекте верхнего уровня на примере современных систем см. в разделе «Дизайн современного ПО». [↑](#footnote-ref-14)
15. Сеймур Паперт – детский психолог, создатель языка LOGO для обучения программированию очень маленьких детей (от пяти лет). [↑](#footnote-ref-15)
16. Бертранд Мейер – создатель языка Eiffel и автор методологии OMT. [↑](#footnote-ref-16)
17. Неправильное понимание (eng.) [↑](#footnote-ref-17)
18. ETH - Eidgenössische Technische Hochschule (dch.) - Высшая государственная техническая школа. [↑](#footnote-ref-18)
19. http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm [↑](#footnote-ref-19)
20. COM – Component Object Model. Технология создания и взаимодействия бинарных компонентов от Microsoft, анонсированная в 1993 году. [↑](#footnote-ref-20)
21. Под Тьюринг эквивалентными вычислительными машинами и языками понимаются такие машины и языки, операционные возможности которых эквивалентны возможностям машины Тьюринга. Доказано, что программа для машины Тьюринга способна реализовать любой вычислительный алгоритм. [↑](#footnote-ref-21)
22. PSPICE – продукт фирмы MicroSim Corporation, принадлежащий целой серии CAE, основанных на вычислительном ядре SPICE Калифорнийского университета в Беркли. Версии PSPICE существуют практически для всех платформ персональных компьютеров и являются стандартом de-Facto для компьютерного моделирования и анализа электронных устройств. [↑](#footnote-ref-22)
23. Shareware - способ распространения программного обеспечения. Деньги выплачиваются только в том случае, если вы хотите поддержать разработку или устали от напоминаний «зарегистрируйтесь, пожалуйста». Распространяемое таким образом ПО обычно целиком разрабатывается одним человеком. [↑](#footnote-ref-23)
24. Доктор Йири Саукап – президент компании Code Farms, бывший директор и один из основателей Cadence Design Systems. Участвовал в проектировании и реализации большого количества CAD систем. Книга «Укрощение C++» описывает проблемы CAD компаний, например Mentor Graphics, при переводе проектов на C++. [↑](#footnote-ref-24)
25. Application Programming Interface (eng.) – Прикладной интерфейс программирования. [↑](#footnote-ref-25)
26. Mainframe – общее название больших (по габаритам) машин эры до появления персональных компьютеров, работа с которыми осуществлялась через терминальный доступ. Машины класса mainframe до сих пор отстаивают право высокопроизводительных систем для бизнеса. [↑](#footnote-ref-26)
27. Дифференцирование уравнений представляет собой не менее рутинную задачу, однако несколько более интеллектуальную и не всегда разрешимую аналитически. [↑](#footnote-ref-27)