Поздняков А.М., руководитель структурного подразделения МОУ «Лицей №124»

Проблемы визуального проектирования.

Визуальное проектирование вместе с более простым способом создания оконного интерфейса к программе и ускорением всей разработки порождает и ряд проблем. Визуальное проектирование возникло на основе и вместе с объектным программированием. Для того, чтобы выявить существенные отличия от процедурного подхода, необходимо рассмотреть структуру программ этих двух вариантов разработки.

Структуру программы при процедурном подходе можно описать блок-схемой. Это будет известная структура с разветвлениями и циклами. Она дает общую схему алгоритма, и принято считать такую схему исходной для разработки программы, хотя заранее для сложных программ ее построить невозможно. Невозможно по той простой причине, что такой схемы до отладки программы просто не существует. Ее можно получить путем проб и поиска ошибок в основной схеме алгоритма на работающей программе. В схеме на бумаге какую-либо ошибку для сложных алгоритмов обнаружить не удастся – можно исходить только из общей, но не детальной схемы, которой является сама программа на языке программирования.. Поэтому основные конструкции алгоритма: последовательность, разветвление, цикл,- это способ мышления при конструировании программы. На основе этих конструкций можно описать задачу любой сложности, а, как их разместить, заранее сказать невозможно – приходится варьировать. Но, если говорить о способе мышления при разработке программ, то еще более важную роль играет блочное представление схемы программы. Фактически любой алгоритм мы можем построить, если использовать последовательность и вложенность блоков (Рис. 1).

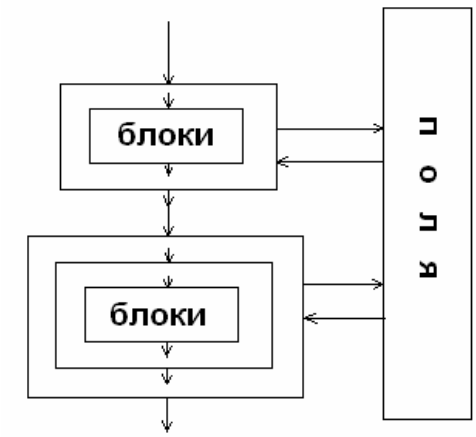

Рис. 1

Основные правила:

1. У блока один вход и один выход.
2. Они могут следовать друг за другом и вкладываться друг в друга.
3. Если мы по структуре программы входим в блок, то обязательно должны из него выйти, а не перепрыгнуть в другой.

Это определенный порядок, используя который можно всегда достичь цели с минимальными затратами. Можно построить и более запутанный алгоритм, игнорирующий эти правила, но такую программу для сложной задачи невозможно отладить. Для более простого введения этого порядка языки программирования имеют

открывающие и закрывающие операторные скобки блоков программы. Блок можно оформить в виде процедуры - легко просматривается процедурный подход.

Объектный подход в принципе не внес в такую структуру существенных изменений. Его главное назначение было объединить в одно целое поля, описывающие свойства какого-либо реального объекта, вместе с процедурами, изменяющими эти поля. Мыслить стало легче – целый блок с множеством процедур и полей на языке может обозначаться одной переменной. Потомки и виртуальные методы только загромождают, но не изменяют этой главной картины.

Данные поступают в программу последовательно. При чтении программы в любой ее точке можно с достаточно большой определенностью сказать, как заполнены поля программы, и, в какой степени решена задача.
Существенные отличия возникли при возникновении визуального проектирования или, можно так назвать, событийного программирования.

Рассмотрим программу с оконным интерфейсом. На экране расположено несколько окон, каждое из которых вносит в данные программы свой вклад. Так как мы можем выполнить какое-либо преобразование и ввод данных в любом окне, то у нас последовательная цепочка оказывается разорванной. И это не просто чисто внешний интерфейс. При визуальном проектировании и оконном интерфейсе любая реакция в окне: нажатие на кнопку мыши, ввод данных с клавиатуры называется событием. События обрабатываются независимыми друг от друга процедурными блоками. Они могут быть связаны друг с другом через внешние поля в общем обвязывающем их объекте, но это не приводит схему программы к предыдущей. Каждое окно может работать без всякой последовательности, произвольно тогда, когда пожелает пользователь. В этом случае алгоритм работы с данными определяется не программой, а пользователем. Если попытаться построить структуру такой программы, то придется ввод данных в поля программы через процедурные блоки расположить параллельно (Рис. 2).

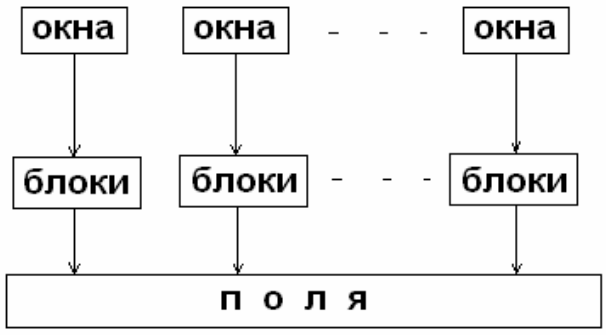

Рис. 2

Если посмотреть на текст программы, то он теперь читается не так, как это было раньше. Процедурные блоки расположены в нем в произвольной последовательности по мере проектирования и невозможно понять, где у программы начало, а где – конец. В оконной задаче так и есть – программа на самом деле не имеет начала – просто возникает задача, позволяющая параллельно вводить данные. Завершение задачи теперь не означает завершение преобразования и вывода данных, данные могут никак не отображаться,- просто завершается задача.

Возникает вопрос – стало ли легче программировать в таком режиме ? Даже простой анализ показывает, что при параллельной структуре увеличивается многообразие возможных вариантов описания задачи. Разработчик и пользователь могут свободнее оперировать блоками, они теперь не связаны жесткой последовательной структурой. Для таких задач теряется формулировка алгоритма как последовательности действий. Кажется, что они выполняются пользователем в какой-либо последовательности, но ее теперь оговорить заранее не всегда удается. Пользователь, работая в произвольном режиме, может допускать ошибочные действия, некоторые должен повторять несколько раз, может быть, он зайдет в тупик и решения не получится. Теперь уже лучше говорить, что он получает не решение, а достигает какой-либо цели, и эта цель может быть достигнута не за один, а за несколько заходов. Может быть, и так, что целей несколько, и некоторые из них достижимы, а некоторые – нет. В программе это заранее не учтешь, в инструкции не отразишь, как при последовательной работе. Научиться достигать цели можно только на опыте, работая с такой задачей.

Как оценить трудоемкость разработки ? Легче ли разработчику иметь дополнительную свободу ? Анализируя способ визуального проектирования можно сразу сказать, что простые задачи при этом проектировать достаточно легко. Система визуального проектирования имеет довольно много средств управления данными и алгоритмом, но при увеличении количества окон - параллельных ветвей ввода данных, - трудоемкость резко возрастает

A B C	F
0 0 0	1
0 0 1	---
0 1 0	1
0 1 1	1
1 0 0	---
1 0 1	0
1 1 0	0
1 1 1	---

Рис. 3

Можно оценить трудоемкость такой разработки.

При 5 окнах у нас 5 вариантов состояния полей, заполнены они или нет. Это значит, что количество всех состояний 2^5=32 варианта. Если нужно что-то сообщить пользователю, то нужно учесть все эти 32 варианта. Возможно, будет меньше – это верхняя оценка. При 10 окнах – 1024 состояния. Такой ситуации нет при последовательной работе, так как по мере выполнения программы в каждой ее части и в конце работы состояние полей точно известно.

Это то новое качество, которое возникает при оконном событийном проектировании. Никаких рекомендаций, как управлять такой ситуацией, визуальные средства проектирования не дают и каких-либо структур не имеют. Что можно рекомендовать для разработки таких программ.

1. ввести таблицу закрытых и открытых окон;
2. ввести таблицу состояний полей в зависимости от возникновения событий связанных с ними окон; фактически это должна быть полная таблица истинности с полным перебором все значений переменных.
3. в этой таблице необходимо задать логическую функцию, в которой нужно отметить те строки, которые являются существенными для

решения какой-либо задачи, а остальные вычеркнуть (не полностью определенная функция. Рис. 3)..

4. каждую отмеченную строку связать с набором действий:

- открытие и закрытие окон;
- сообщение о правильных или неправильных действиях пользователя;
- анализ состояния всей задачи.

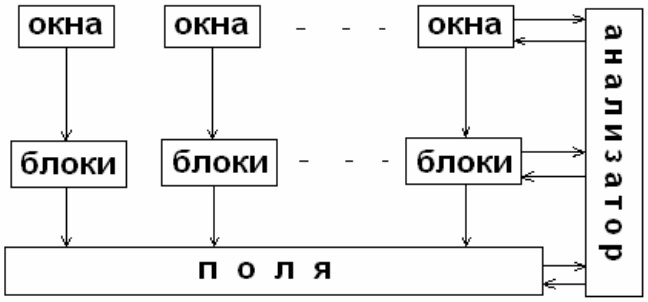

Рис. 4

Самой важной частью является анализ состояния. Это дополнительная нагрузка на разработчика, которой нет при последовательной работе, и без которой такие задачи трудно использовать.

В анализ состояния должна входить оценка, что реализовано, и, что можно предпринять дальше (Рис. 4).

Казалось бы, это только трудности программиста, но параллельный вход в задачу имеет более далеко идущие выводы. Возникает класс задач, который из-за параллельного входа, более приближен к реальности, более похож на действия человека.

В этих задачах есть вид памяти, которая может заполняться по мере выполнения каких-либо действий. Программа проявляет некоторые первичные признаки интеллекта, анализируя состояние памяти, и, сообщая об этом пользователю.

Как и раньше, ее конструкция и идея зависят от разработчика, но ее структура устроена по-другому. В теле программы есть отдельные последовательные блоки, расположенные в параллельных ветвях. Их параллельную работу определяет некоторое ядро программы - центральный анализатор. Вся задача или связанный с ней объект может изменяться по мере работы параллельных ветвей и данных в полях от простого, исходного состояния к более сложному.

У объекта может быть несколько конечных состояний. При большом количестве параллельных ветвей некоторые могут быть непредсказуемы. Теряется понятие четкости и конечности алгоритма. Некоторых состояний объекта можно достичь, не выстраивая логических построений, а на уровне ассоциаций.