УДК 004.51

# РАЗРАБОТКА ПРИЛОЖЕНИЙ С ГИБКИМ ИНТЕРФЕЙСОМ КАК СРЕДСТВО ПРИВЛЕЧЕНИЯ И УДЕРЖАНИЯ ПОЛЬЗОВАТЕЛЕЙ

**Мыльников А.А., Личаргин Д.В.**
**научный руководитель канд. техн. наук Личаргин Д.В.**
***Сибирский Федеральный Университет***

В статье рассматривается решение проблемы использования унифицированного интерфейса пользователей, а так же модель разработки приложений с гибким интерфейсом.

Проблема унифицированных интерфейсов является актуальной в связи с выпуском различных версий графических приложений. Данная проблема решается на стыке таких наук, как информатики, теории компиляторов, теории алгоритмов и т.д.

Вопрос о наличии гибких настроек интерфейсов требует исследований в рамках вопроса предоставления пользователю режима доступа к изменению проекции иерархии внутренних данных программы (баз данных, объектов интерфейсов на различные формы представления данных).

Цель данной работы состоит в постановке проблемы построения максимально гибкого, редактируемого пользователем интерфейса программных систем.

Задачи данной работы:

1. Построение модели программного продукта как мультииерархичной системы.
2. Рассмотрение проекции мультииерархии данных программного продукта на конкретные формы представления данных (поле memo, поля edit, выпадающие списки и т.д.).
3. Предложен способ решения этой задачи без компиляции с точки зрения методики программирования.

Новизна работы состоит в использовании модели вариативной мультииерархии основанной на теории графов и теории классификации.

Отцом GUI (GUI - graphical user interface) считается Дуглас Карл Энгельбарт (Douglas Karl Engelbart), опубликовавший в 1962 году свои идеи в докладе “Усиление человеческого интеллекта”. Это был огромный скачек в мышлении на 1962 год, в связи с чем, доклад нашел поддержку и Дуглас получил финансирование. Собрав группу ученых, Энгельбарт со своей командой разработали элементы компьютерного интерфейса, такие как «мышь», «гипертекст» и задатки графического интерфейса.

В последующем, над разработкой GUI работало множество компаний, среди которых самые значительные были Xerox и Apple, положившие старт новой эре инноваций в компьютерной графике.

Таким образом,то, что раньше было фантастикой (а именно отказ от использования перфокарт, ввод данных с терминала или же вывод в терминал результата), сейчас обычное дело. Однако разработки и идеи не останавливаются.

**Проблема унифицированных GUI.**При разработке графического интерфейса пользователя, разработчик(и) опираются на личное мнение дизайнеров. Однако дизайнер не может предусмотреть мнение всех пользователей относительно приложения, в связи с чем, всегда находятся пользователи недовольные текущим GUI. И если имеется альтернатива приложению, пользователь станет искать более приемлемый для него в использовании аналог. Концепция DWIM (Do What I Mean – «делай то, что я имею в виду») зачастую не соблюдается в современных приложениях. Дизайнерам, привыкшим к современным стандартам графических приложений,кажется

вполне логичным определенные иконки, названия или же поведение элементов, что зачастую расходится с мнением пользователей.

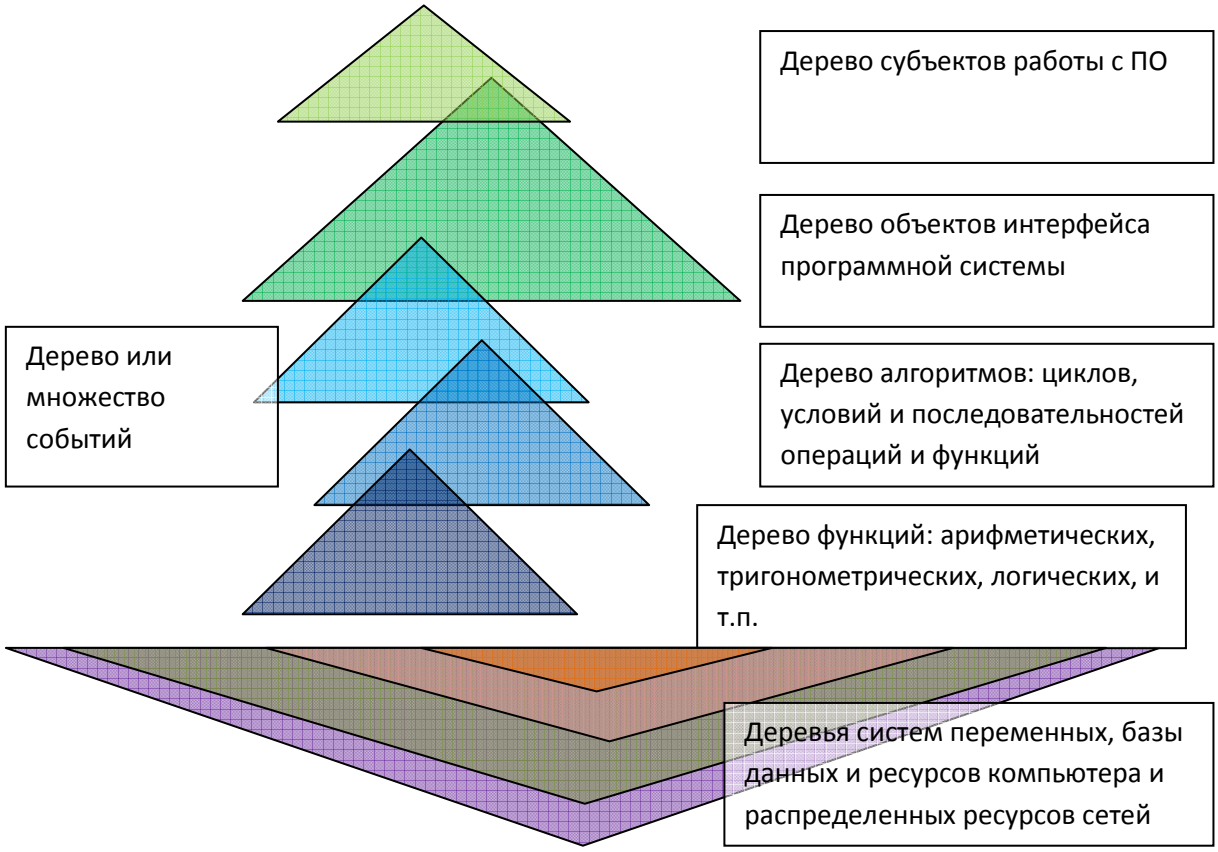


Рис. 1. Мультииерархия программного обеспечения

Гибкий GUI,это не отказ от огромной работы дизайнеров, это самая легкая возможность удержания и привлечения пользователя. Если пользователю что-то не нравится в дизайне, он волен сам подстроить его под себя.

Большинство приложений позволяет изменить горячие клавиши, что является весьма удобным, как и изменения панели быстрого доступа.

Но можно дать пользователю возможность не только изменить панель быстрого доступа, но и все остальные элементы подстроить под себя. Однако, вспоминая Unix и X Windows System, можно пойти и дальше, соединяя в интерфейсе одного приложения, функции другого приложения.

**API (Application Programming Interface).** В такой модели, API следует реализовать с терминальным доступом, причем с возможность обработки вводимой команды, а так же с базой данных API. Таким образом, любое приложение сможет создать процесс с нашим приложением, а используя базу данных получить информацию обо всех доступных командах. Приведем следующий пример. Возьмем некий файловый менеджер, который переходит по каталогам и отображает их содержимое. Для каждого из этих действий, он использует команду операционной системы, а результат после обработки выводит в графическом виде. При этом операционная система не использует меню(1 - перейти, 2 - вывести, и т.д.), а просто обрабатывает команду и (если необходимо) выдает результат.

Рассмотрим реализацию GUI. Существуют инструменты, с помощью которых можно довольно просто использовать данную модель. Один из них – декларативный

язык программирования QML. QML очень похож на XML (eXtensible Markup Language — расширяемый язык разметки), поэтому можно создавать с помощью QML все окна приложения совместно с дизайнером, а потом позволить пользователю добавлять, удалять или же изменять элементы текущего окна. После сохранения пользователем, приложению всего лишь необходимо выбрать дизайн окна пользователя за место заданного по умолчанию.

Таблица 1.
Типы деревьев составляющих мультииерархическую систему программного обеспечения

| **Типы объектов и явлений** | **Пример** |
|---|---|
| Дерево объектов интерфейса | Form1<br>Pannel1<br>Table1<br>Pannel3<br>Button1<br>Pannel2<br>Image1<br>Memo1 |
| Дерево подобъектов | Market.Buyer.Taking.WhatIsTaken.PlaceToHold.Form.<br>Colour |
| Дерево алгоритмов (условий, циклов и последовательностей) в методах | For … do<br>Begin<br>If … then<br>Beginelse<br>For … do<br>Begin<br><br>End;<br>For … do<br>Begin<br><br>End;<br>End;<br>End; |
| Дерево функций | Sum(<br>3,<br>Multiply(<br>Sum(<br>4,<br>Minus(<br>4,<br>Sin(<br>5))))) |
| Дерево или массивы параметров | [[1,4],[ 2,6], [3,5],[1, 8]] |
| Значения параметров | X = 6.78, Y = «love» |

**Использование порождающих грамматик над деревьями строк.** Как известно, стандартные порождающие грамматики над строками имеют вид четверки:

G<S, T, N, R>, где S – начальный символ порождающей грамматики, T – множество терминальных символов, N – множество нетерминальных символов, и R – множество правил трансформации одной строки в другую.

Для порождающих грамматик над деревьями строки символов t и n заменяются деревьями (или лесом – деревьями с тождественными узлами). t = t<t', t'',…,$t^n$>, где t' = t' <$t^1$, $t^2$,…,$t^m$> и т.д., n = n<n', n'',…,$n^n$>, где n' = n' <$n^1$, $n^2$,…,$n^m$> и т.д.

Порождающая грамматика над деревьями строк строится следующим образом. Пусть A<…B<…C1→C2…>,…, B'<…C1'→C2'>…> – правило порождающей грамматики над деревьями из множества таких правил с деревьями строк терминальных символов T и нетерминальных символов N, «→» - символ перехода одной строки в другую. S<> – начальный символ порождающей грамматики над деревьями. Углубление дерева состояний другого генерируемого дерева или леса строк состоит на каждом этапе в умножении получаемого генерируемого дерева на правило порождающей грамматики.

Возможно применение порождающих грамматик над деревьями строк (таких как грамматики Монтегю) в целях генерации кода обеспечивающего автоматическое добавление в программу функций гибкого интерфейса.

Таким образом, при обработке определенных объектов, например, закладок, можно будет порождать списки альтернатив этих объектов, например, панелей, дополнительных форм, панелей меню и выпадающих списков со ссылками на другие формы, закладки, панели, выпадающие меню. Что может дать возможность трансформировать интерфейс в сторону максимальной гибкости.

Возможна договоренность использовать опцию «Flax» - гиб[кий интерфейс], для стандартизации входа в раздел настройки гибкого интерфейса для произвольных программ. В этом разделе необходимо отобразить деревья интерфейса и множества / деревья событий (см. Рисунок 1, см. Таблицу 1). Можно осуществлять выбор вариантов замены узлов дерева на эквивалентные по структуре данных элементы или кластеры элементов, преобразовывать деревья данных программного обеспечения на основе операций: копировать, вырезать, вставить, осуществлять загрузку и настройку параметров форм: ширины, высоты, цвета, фактуры, изображения, форм в гибком режиме. Предполагается не допускать редактирование деревьев функций и деревьев алгоритмических операций: циклов, условий и последовательностей пользователем с точки зрения общих принципов безопасности программной системы.

На основе такого простого редактора встроенного в произвольную программу(посредством расширения и компиляции кода встроенными модулями языков программирования с использованием расширенных порождающих грамматик Монтегю) можно было бы копировать файлы настроек интерфейса своих друзей, известных людей из социальных сетей, работающих ради компьютерного сообщества дизайнеров, художников и просто энтузиастов. Использование модели гибкого интерфейса, может послужить хорошей возможностью удержать старых пользователей при выпуске новой версии продукта с графическими изменениями в приложении, а так же привлечь новых пользователей, которых не устраивает GUI конкурента.

**Выводы.** Предложена модель гибкого интерфейса программных систем на основе мультииерархичной проекции программного обеспечения на списки эквивалентных форм интерфейсного доступа к данным. Были рассмотренывозможностиданной системы на основе компилируемых и интерпретируемыхпрограмм. Подчеркивается важность продолжения исследований по теме создания гибких интерфейсов на основе наиболее полной автоматизации этих процессов.