3. *Аверкин А.Н., Батырин И.З., Блиншун А.Ф., Силаев Б.В., Тарасов Б.Н.* Нечеткие множества в моделях управления и искусственного интеллекта. - М.: Наука, 1986. -312 с.
4. *Финаев В.И., Белоглазов Д.А.* Микропроцессорный нечеткий регулятор подачи топлива // Материалы VII Всероссийской научной конференции студентов и аспирантов «Техническая кибернетика, радиоэлектроника и системы управления». – Таганрог, 2004.
5. *Заде Л.* Понятие лингвистических переменных и его применение к принятию приближенных решений. – М.: Мир, 1976. – 165 с.

**Белоглазов Денис Александрович**
Технологический институт Федерального государственного образовательного учреждения высшего профессионального образования «Южный федеральный университет» в г. Таганроге.
E-mail: d.beloglazov@gmail.com.
347928, г. Таганрог, пер. Некрасовский, 44.
Тел.: 88634371689.

**Коберси Искандар Сулейман**
E-mail: salouma1@mail.ru.

**Финаев Валерий Иванович**
E-mail fin_val_iv@tsure.ru.

**Beloglazov Denis Aleksandrovich**
Taganrog Institute of Technology – Federal State-Owned Educational Establishment of Higher Vocational Education “Southern Federal University”.
E-mail: d.beloglazov@gmail.com.
44, Nekrasovskiy, Taganrog, 347928, Russia.
Phone: 88634371689.

**Kobersy Iskandar Syleiman**
E-mail: salouma1@mail.ru.

**Finaev Valery Ivanovich**
E-mail fin_val_iv@tsure.ru.

УДК 004.416.6

**И.А. Колоколов, А.Н. Литвиненко**

## АДАПТИРУЕМОСТЬ ПРОГРАММЫ К МОДИФИКАЦИЯМ

*Рассматривается важное свойство программы – "адаптируемость к модификациям" и методы его достижения. Вводится понятие "виртуальные однородные пространства". Использование однородных пространств и регулярных соглашений позволяет существенно повысить "адаптируемость к модификациям" программ за счет применения ассоциативных связей и дополнительной косвенности.*

*Адаптируемость; безболезненность; однородное пространство; виртуальное однородное пространство; регулярные правила; инвариантность; ассоциативные связи; метапрограммирование.*

**I.A. Kolokolov, A.N. Litvinenko**

## PROGRAM ADAPTABILITY TO MADIFICATIONS

*The paper studies the important property of program - "adaptability to modifications" and methods of it achievement. The concept of "virtual homogeneous spaces" is introduced. Using of homogeneous spaces and regular agreements allows greatly to raise the "adaptability to modifications" of programs due to applying associative relations and additional indirection.*

*Aadaptability; painlessness; homogeneous space; virtual homogeneous space; regular rules; invariance; associative relations; metaprogramming.*

Любая крупная программа (семейство программ) в своем жизненном цикле претерпевает революционные и эволюционные изменения. Революционные изменения кардинально преобразуют программу, однако потребность в них возникает достаточно редко. Основной объем программистских усилий приходится, как правило, на эволюционные изменения [6]. Эволюция программы идет частыми, но относительно небольшими шагами. На каждом шаге имеющийся код меняется (добавляется/удаляется/изменяется некоторая функциональность). Как сделать так, чтобы новые изменения органично вписались в уже существующую программу и чтобы эти эволюционные изменения проходили безболезненно, не угрожая работоспособности ранее написанных частей программы? Ответ на данный вопрос дается в данной статье.

Сформулируем основную ***задачу***: необходимо повысить надежность (безболезненность) изменений и уменьшить требуемые ресурсы (временные, людские, требования к квалификации) при внесении эволюционных модификаций в программу.

**Проблема внесения эволюционных изменений в сложную программу.** Сегодня опытному программисту, да и вообще любому программисту, достаточно редко выпадает шанс начать работу над проектом "с чистого листа". В основном же, большая часть программистов трудится над сопровождением сформировавшихся или разработкой почти сформировавшихся программных продуктов. Но, когда выпадает этот шанс, программист полон оптимизма, что в этот раз он напишет "правильную" программу, реализует в ней все свои замыслы и правила, выкристаллизовавшиеся с опытом, для того чтобы возможные дальнейшие модификации его программы не нарушали ее работоспособности. И, напротив, как мучается программист, когда, достигнув некоего предела сложности своей программы, он уже не может без содрогания и раздражительности слышать о необходимости внесения в нее новых изменений или дополнений, поскольку в памяти у него свежи воспоминания о небольших изменениях программы, приведших к ее некорректной работе или даже к ее неработоспособному состоянию на несколько дней, вследствие чего заказчик потеряет n-ю сумму денег. Но делать нечего, заказчик ждет, вносить изменения все равно придется, и тогда программист вопреки своему желанию и своим принципам идет к компьютеру, вносит все необходимые изменения ... и надеется, что ничего "жизненно" важного не затронул в "организме" программы, что может привести к нарушению работоспособности.

Пока размер вашей программы невелик и исчисляется несколькими файлами, вы можете даже и не задумываться о возможных проблемах и сложностях ее сопровождения. Но когда ваша программа состоит из нескольких сотен или тысяч файлов со сложной логикой и структурой, вы начинаете осознавать, что теряете контроль над программой и что уже не программа работает на вас, а вы на программу. Развитие любого программного проекта немыслимо без редактирования. Нередко эволюционные изменения в программе приводят к нарушению ее работоспособности, но самое страшное заключается в том, что большая часть программистов считает, что внесение очередного эволюционного изменения в программу без редактирования ранее отлаженного исходного кода либо вообще невозможно, либо для этого нужны какие-то немыслимые усилия и жертвы со стороны программиста, на которые он не готов идти.

**Механизмы, обеспечивающие "адаптируемость к модификациям".** Существует ряд механизмов, предназначенных для обеспечения надежности внесения очередного изменения в программу:

- *Модуляризация* с использованием цепочечного или каркасного подхода – расчленение программного материала на модули с цепочечной или каркасной организацией [5].
- *Порождающее программирование* (*generative programming*) – парадигма технологии разработки программного обеспечения, основанная на моделировании семейства программных систем, используя которые можно по конкретным техническим требованиям автоматически получить специализированный и оптимизированный промежуточный или конечный программный продукт из элементарных многократно используемых компонентов реализации с помощью базы знаний о конфигурациях [4].
- *Аспектно-ориентированное программирование (Aspect-oriented programming)* – парадигма программирования, основанная на идее разделения функциональности, особенно сквозной функциональности, для улучшения разбиения программы на модули, которые называются аспектами (aspects) [1].

В данной статье предлагаются новые механизмы для обеспечения программы важнейшим свойством – "адаптируемости к модификациям" и показывается плодотворность применения предлагаемого подхода для сопровождения работающих приложений.

***Основной тезис*** данной статьи: чтобы писать программы, адаптируемые к модификациям, необходимо мыслить и программировать в стиле "для всех сущностей, удовлетворяющих некоторому условию", т.е. выделять и использовать однородные пространства вместо явного перечисления (в том или ином виде) обрабатываемых элементов. Кроме того, нужно вносить в программирование больше регулярности.

**Понятия безболезненности изменений и однородного пространства.** Приведем несколько ключевых определений, которые помогут нам в дальнейшем лучше ориентироваться в рассуждениях и примерах данной статьи.

***Безболезненность*** внесения очередного изменения – внесение очередного изменения в программу называется безболезненным, если после его внесения не нарушается работоспособность всей программы (функций и свойств, существовавших до внесения изменения) [5, 6].

***Однородное пространство*** – это набор однотипных элементов, имеющих одинаковые свойства и методы обработки. Элементы могут обладать достаточно сложной структурой и логикой. Однородное пространство имеет две составляющие: само *хранилище* однородных элементов (данные) и *механизм обработки* данного хранилища (процедуры обработки), инвариантные относительно количества элементов в хранилище этого пространства.

Элементы называются ***однотипными***, если они принадлежат одному и тому же семейству и решают один и тот же класс задач. Например: индексы таблицы, пункты меню, операции по правой кнопке и т.д.

***Постоянное однородное пространство*** – это однородное пространство, хранилище которого представляется созданной заранее структурой данных, содержащей некоторые элементы. Чаще всего оно имеет конкретное физическое представление на жестком диске в виде файла, таблицы и т.д. Постоянство является характеристикой хранилища однородных элементов. Например, таблица, содержащая список всех операций по правой кнопке мыши проекта. В данном случае таблица – это однородное пространство, а операции по правой кнопке мыши – это однотипные (однородные) элементы.

***Виртуальное однородное пространство*** – однородное пространство, которое заранее не задано и всегда собирается (строится) в динамике по определенным

правилам в результате выполнения некоторого "*запроса построения однородного пространства*". Результат такой сборки может храниться как на жестком диске, так и в виртуальной или оперативной памяти. Виртуальность является характеристикой хранилища однородных элементов. Например, курсор (являющийся результатом запроса SELECT), содержит список всех индексов базы данных. В данном случае курсор – однородное пространство, а индексы – это однотипные (однородные) элементы.

Элементы однородного пространства могут быть разных типов:

- *декларативные* – элементы, содержащие только декларативную информацию (определение каких-нибудь параметров или переменных);
- *процедурные* – элементы, содержащие только процедурную информацию (выполнение каких-нибудь операций);
- *смешанные* – элементы, содержащие одновременно и декларативную и процедурную информацию.

Одним из важных атрибутов элемента однородного пространства является "*условие применимости*" данного элемента, причем для виртуальных однородных пространств это условие может быть вынесено на этап построения самого пространства. Например, имея однородное пространство пунктов меню, часть пунктов может отсутствовать в зависимости от какого-нибудь условия, например, под каким пользователем был произведен вход в программу.

Самые распространенные типы хранилища для однородных пространств – это таблицы (или курсоры) и XML-файлы (или строковые переменные с XML-структурой), но могут быть и другие, например текстовые файлы со специальной разметкой, массивы и т.д.

Постоянные однородные пространства программист создает заранее руками или с помощью специальных запросов, т.е. они всегда предопределены, а виртуальные однородные пространства всегда создаются в динамике с помощью специальных "запросов построения однородного пространства". ***Запросы построения однородного пространства*** – механизмы (программы), результатом работы которых является хранилище однородного пространства. Их можно реализовать разными способами, с помощью языка SQL (для обработки таблиц), XQuery или XSLT [7] (для обработки XML-файлов), или же с помощью процедур и функций целевого языка программирования (для обработки любых текстовых файлов). Принцип работы таких "запросов" заключается в том, что они делают выборку однотипных объектов из программного фонда проекта (*программный фонд* – совокупность программ, данных, тестов, документов и других материалов, накапливаемых при реализации программного проекта [5]), удовлетворяющих определенному условию (условие указывается в запросе), и результатом выполнения "запроса" является хранилище однородного пространства, содержащее набор однотипных элементов. Например, для получения однородного пространства, содержащего набор индексов таблиц Table1 и Table2 в качестве однотипных элементов с атрибутами: имя таблицы, имя индексируемой колонки и имя индекса, необходимо написать следующий SQL запрос:

```
Select object_name(i1.object_id) As tablename,
        col_name(i1.object_id,i1.column_id) As colname,
        i2.name AS indexname
 Into #indexinfo
 From sys.index_columns i1
  Inner Join sys.indexes i2 On i1.object_id=i2.object_id and
i1.index_id=i2.index_id
 Where object_name(i1.object_id) In ('Table1','Table2')
```

Результатом данного запроса будет временная таблица #indexinfo (виртуальное однородное пространство), содержащая информацию об индексах таблиц Table1 и Table2. И далее с этой временной таблицей мы можем работать как с однородным пространством, применяя механизмы обработки данного пространства. Например, можно удалить эти индексы, можно их переименовать или сменить их режим сортировки и т.д. Причем добавление/удаление/изменение индекса в одной из таблиц Table1 или Table2 не влечет за собой изменения ранее отлаженных фрагментов кода, индекс автоматически добавится/удалится/изменится в получаемом однородном пространстве. Создавая данное однородное пространство, мы вынесли обработку "*условия применимости*" (берем только индексы таблиц Table1 и Table2) на этап построения самого пространства.

Ключевым этапом проектирования/разработки любой крупной системы является работа с однородными пространствами. Важно вовремя увидеть, сформировать и применить однородное пространство. Некоторые однородные пространства видны сразу и очевидны, а некоторые мы замечаем позднее. Почему мы не сразу видим однородность? Это происходит из-за того, что со временем наша программа может расти, обрастать новой логикой и новыми механизмами реализации этой логики, в связи с этим могут появляться однотипные элементы системы, количество которых может определить появление однородного пространства.

Однородные пространства – это своего рода конфигурационный ориентир [5]. Такой же, как выделение объектов в объектно-ориентированном программировании [2], выделение аспектов в аспектно-ориентированном программировании [1], вертикальные и горизонтальные слои Фуксмана [3] – это все конфигурационные ориентиры. Исходя из определения конфигурационного ориентира, данного Горбуновым-Посадовым М.М. в книге "Расширяемые программы" [5], ***конфигурационный ориентиры*** – это решения и механизмы, к которым априорно тяготеет программист при разработке программы.

**Адаптируемость и инвариантность**

***Инвариантность*** (лат. invariabilis – неизменный) – неизменность, независимость от некоторых условий или по отношению к некоторым преобразованиям.

***Адаптируемость к модификациям*** можно определить как возможность внесения в программу безболезненных эволюционных модификаций. Важно отметить, что "адаптируемость к модификациям" – это не только внесение эволюционных модификаций без нарушения работоспособности программы, но и возможность внесения изменений без редактирования ранее отлаженных работоспособных фрагментов программы (инвариантность программы относительно изменений). Таким образом, адаптируемость и инвариантность тесно связаны между собой, а именно, понятие "адаптируемости к модификациям" включает в себя понятие инвариантности.

Например, есть СУБД-приложение и в этом приложении есть таблица организаций, где хранится информация об организациях (наименование, адрес, телефон и т.д.), то возникает необходимость в добавлении нового атрибута "ИНН" для организаций, которая влечет за собой внесение изменений в следующих местах: в таблице организаций; в обновляемых курсорах, в определении которых участвует таблица организаций; в экранной форме, реализующей справочник организаций; в отчетах по организациям. То есть приходится вносить изменения в ранее отлаженные работоспособные фрагменты программы, а это может повлечь за собой нарушение работоспособности программы. В идеале, необходимы механизмы, которые позволяли бы не редактировать ранее отлаженный программный код при появлении нового атрибута "ИНН" у организаций, и таким механизмом являются однородные пространства.

Напомним еще раз ***основной тезис*** данной статьи: чтобы писать программы, адаптируемые к модификациям, необходимо мыслить и программировать в стиле "для всех сущностей, удовлетворяющих некоторому условию", т.е. выделять и использовать однородные пространства вместо явного перечисления (в том или ином виде) обрабатываемых элементов.

**Регулярность**

Часто, чтобы было возможно выделять однородные пространства, необходимо вносить в программирование больше регулярности.

Регулярность бывает нескольких видов:

- ***регулярность в наименованиях*** – определенные правила в наименованиях переменных, процедур, функций, файлов, индексов, связей (relation), свойств объектов и так далее;
- ***регулярность в ограничениях*** – определенные правила, следуя которым нельзя удалить/добавить/изменить какие-то части программного фонда системы. Например, ограничение целостности в базах данных;
- ***регулярность в умолчаниях*** – определенные правила, которые выполняются всегда по умолчанию, если разработчик их не переопределил или же вообще всегда по умолчанию без доступа переопределения для программиста. Яркий пример такой регулярности – это наследование в объектно-ориентированном программировании.

Используя больше регулярности в программе, нам, во-первых, будет проще разбираться в программе, а во-вторых, мы можем строить однородные пространства однотипных элементов, удовлетворяющих каким-то регулярным правилам.

Таким образом, регулярность помогает в определении однородных пространств, а значит, помогает повысить адаптируемость к модификациям программы.

**Примеры, иллюстрирующие адаптируемость к модификациям.** Сначала рассмотрим простейший пример "Select * From", который повышает свойство "адаптируемость к модификациям" программы.

Пример "Select * from"

В языке SQL существует конструкция:

*select * from mytable* (а)

позволяет сделать выборку всех столбцов из указанной таблицы mytable.

Существует и альтернативный способ выборки всех столбцов из таблицы, это достигается путем перечисления наименований всех столбцов таблицы:

*select column1,column2,...,columnN from mytable* (б)

Таких фрагментов, как (а) и (б), в программе может быть n-е количество, переплетенных с другими фрагментами, совместно реализующих некий функционал. Добавление нового столбца в таблицу mytable или удаление старого повлечет за собой изменение всех фрагментов типа (б) (при этом надо найти все такие фрагменты, а это источник потенциальных ошибок, так как можем забыть изменить какой-либо фрагмент типа (б)), а фрагменты типа (а) останутся без изменения.

Таким образом, использование конструкции "*" вместо явного перечисления заметно повысит скорость и надежность эволюционных модификаций, т.е. повысит адаптируемость программы к модификациям.

**Пример с заменой нескольких элементов справочника на заданный.** В СУБД-приложениях часто возникает потребность в замене одних данных на другие с сохранением целостности базы данных. Рассмотрим один из таких при-

меров. Имеется справочник организаций, необходимо удалить из него организации "ПромСтрой" и "Пром-Строй", заменив при этом все ссылки на данные организации ссылкой на организацию ООО "ПромСтрой" во всех таблицах базы данных, связанных со справочником организаций. При этом целостность базы данных не должна нарушиться.

Данная задача сводится к выделению множества "всех вхождений ссылок на заменяемые организации по всей базе данных" (такие вхождения могут быть в разных строках разных таблиц) и к действию по замене этих ссылок на нужную ссылку.

Один из вариантов построения данного множества – это явное перечисление всех вхождений ссылок на заменяемые организации по всей базе данных. Но это «путь в пропасть», так как появление новых таблиц (колонок) или удаление старых таблиц (колонок) со ссылкой на заменяемые организациями в базе данных всегда будет вести к редактированию фрагмента программы, связанного с построением множества "всех вхождений ссылок на заменяемые организации по всей базе данных". То есть необходимо будет изменять ранее написанные и отлаженные фрагменты программного кода, а если этого не делать, то будут возникать ошибки на этапе обработки полученного множества. Таким образом, адаптируемость программы к модификациям не достигается.

Следуя принципу адаптируемости программы к модификациям (чтобы писать программы, адаптируемые к модификациям, необходимо мыслить и программировать в стиле "для всех сущностей, удовлетворяющих некоторому условию", т.е. выделять и использовать однородные пространства вместо явного перечисления (в том или ином виде) обрабатываемых элементов), необходимо в качестве множества "всех вхождений ссылок на заменяемые организации по всей базе данных" использовать виртуальное хранилище однородного пространства, построенное с помощью SQL-скрипта (учитывая связи между таблицами), и работать уже напрямую с этим пространством. Таким образом, появление новых записей или удаление старых записей с заменяемыми организациями в базе данных всегда автоматически будет учитываться в динамике, и не нужно будет каждый раз менять ранее написанные механизмы, решающие проблему с заменой вхождения одних данных на другие. А это и есть адаптируемость программы к модификациям.

Таким образом, использование виртуальных однородных пространств вместо явного перечисления обрабатываемых элементов делает программу инвариантной относительно данного класса задач.

Из рассмотренных примеров можно сделать **вывод**, что использование однородных пространств вместо явного перечисления обрабатываемых элементов повышает скорость внесения эволюционных изменений и дает возможность производить эволюционные изменения без нарушения работоспособности программы и без редактирования ранее отлаженных работоспособных фрагментов программы, т.е. повышает адаптируемость программы к модификациям.

**Безболезненные модификации с помощью ассоциативных связей**

***Ассоциативные связи*** – способы неявной связи программных компонент. Эти способы обеспечивают связь фрагментов программного кода, реализующих некоторую модификацию, с остальной программой без явного задания ссылок на фрагменты, реализующие эту модификацию. Модификация локализуется строго в одном отдельном файле, а связь с остальной программой происходит через ассоциативные механизмы по некоторым регулярным правилам или через дополнительную косвенность. При этом отпадает необходимость изменять текст программы, существующий до внесенного изменения. Ассоциативные связи позволяют

выделять сквозную функциональность в отдельные модули. ***Сквозная функциональность*** – функциональность, реализация которой разбросана по различным модулям программы. Сквозная функциональность приводит к рассредоточенному и запутанному программному коду. *Запутанным* называется такой код, в котором одновременно реализована различная функциональность. *Рассредоточенный код* – это такой код, когда реализация одной функциональности распределена по разным местам программного кода.

Приведем несколько ассоциативных механизмов связи фрагментов изменений с остальной программой:

- *однородные пространства* – хранилище однородного пространства само по себе обладает необходимой ассоциативностью, так как любое изменение в хранилище моментально отражается в программе по определению принципов работы самого однородного пространства;
- *регулярное описание (правило) определения мест вставки модификации* - все модификации осуществляются в отдельных модулях, которые по определенным регулярным правилам подключаются к основной программе. Например, в начале и в конце некоторой функции выполняем содержимое программных модулей, наименование которых построено по следующему регулярному правилу: Begin_<имя файла>_<имя функции> или End_<имя файла>_<имя функции>. Для этого необходимо реализовать соответствующий ассоциативный механизм и придерживаться необходимых регулярных правил;
- *аспектно-ориентированное программирование (Aspect-oriented programming)* – парадигма программирования, основанная на идее разделения функциональности, особенно сквозной функциональности, для улучшения разбиения программы на модули, которые называются аспектами (aspects) [1].

**Обсуждение и анализ предлагаемого подхода.** Конечно же, чтобы программировать в стиле "адаптируемости к модификациям", необходима более высокая квалификация программиста и дополнительные временные затраты на написание такого первичного кода, которые немного, а иногда и намного (все зависит от задачи) превосходят временные затраты, потраченные на программирование с помощью явных перечислений. Но, тем не менее, это плодотворно влияет на дальнейшее сопровождение программных продуктов (рис. 1).

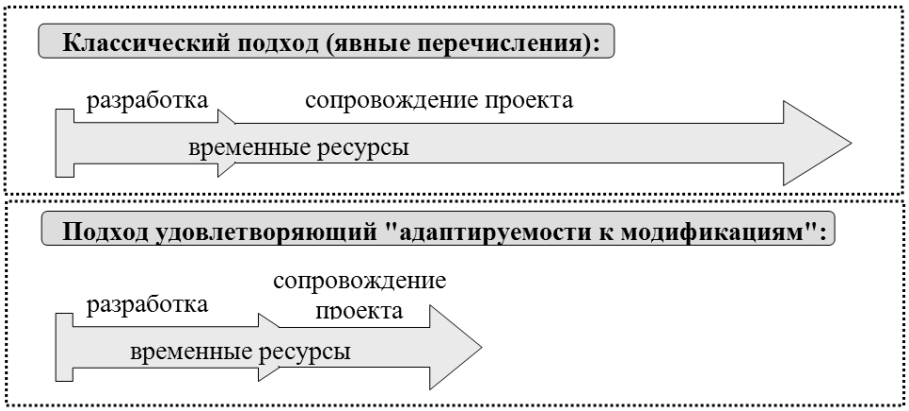


*Рис. 1. Временные затраты разработки и сопровождение проекта*

Необходимо отметить, что дополнительные усилия на разработку специальных механизмов, повышающих адаптируемость программы к модификациям, нужно проделать один раз, а сопровождение программы происходит многократно, на протяжении всего жизненного цикла программы.

Схематически подход с использованием однородных пространств показан рис. 2.


*Рис. 2. Подход с использованием однородных пространств*

Здесь метаданные – это хранилище однородного пространства, метапрограмма – механизм обработки данного хранилища (процедуры обработки), инвариантный относительно количества элементов в хранилище этого пространства и программа – это результат взаимодействия хранилища и механизмов его обработки. Это своего рода метапрограммирование. ***Метапрограммирование*** – создание программ, которые создают другие программы как результат своей работы (в частном случае, программы, изменяющие или дополняющие себя во время выполнения).

Ключевым методом повышения адаптируемости программы является выделение и использование однородных пространств как одного из конфигурационных ориентиров.

А путями достижения этой цели являются: внесение в программу большей регулярности, использование дополнительной косвенности при ссылках на программные объекты, увеличение относительной доли метаданных, превращение программы в "структуру данных", которая обрабатывается метапрограммами.

***Ключевая идея*** использования однородных пространств состоит в замене явного перечисления (в том или ином виде) обрабатываемых элементов, введении дополнительной косвенности, т.е. вводится хранилище однородного пространства и используются методы обработки данного однородного пространства, инвариантные относительно количества элементов в хранилище этого пространства.

Однородные пространства фактически являются местами возможных эволюционных расширений программы. Чем больше таких мест мы предусмотрим при создании программы, тем проще и надежнее будет дальнейшее ее сопровождение.

Основным преимуществом при использовании данного подхода является то, что можно построить программную систему, эволюционные модификации которой будут производиться безболезненно и технологично, а также без редактирования ранее отлаженных, работоспособных фрагментов программы.

**Заключение.** Основные результаты, полученные авторами данной работы, их новизна и отличие от результатов, полученных другими авторами:

- Предложены способы повышения адаптируемости и инвариантности программы относительно широкого класса эволюционных модификаций.
- Впервые введено понятие ***виртуального однородного пространства***, обоснована обязательность атрибута с условием применимости для элементов любого однородного пространства.
- Использование виртуальных однородных пространств позволяет применять ассоциативную (неявную) схему подключения элементов в однородные пространства.
- Показано удобство использования регулярных правил для выделения однородных пространств.

- Предлагается вместо явного перечисления использовать однородные пространства.
- На примерах показана плодотворность использования предложенного подхода для повышения адаптируемости и инвариантности программы.

Достигнута основная задача: повышение надежности (безболезненности) и уменьшение требуемых ресурсов (временные, людские и требования к квалификации) при внесении эволюционных модификаций в программу.

Используя изложенный подход к программированию, можно построить программную систему, эволюционные модификации которой будут производиться безболезненно и технологично, без редактирования ранее отлаженных работоспособных фрагментов программы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Kiczales G., Lamping J., Mendhekar A., etc.* Aspect-oriented programming. Published in proceedings of the European Conference on Object-Oriented Programming (ECOOP). Finland, Springer-Verlag LNCS 1241. June 1997. URL: http://www.oberon2005.ru/paper/gk1997.pdf (дата обращения 10.08.2009)
2. *Фридман А. Л.* Основы объектно-ориентированной разработки программных систем. – М.: Финансы и статистика, 2000. – 192 с.
3. *Фуксман А.Л.* Технологические аспекты создания программных систем. – М.: Статистика, 1979. – 184 с.
4. *Чарнецки К., Айзенекер У.* Порождающее программирование: методы, инструменты, применение. – СПб.: Питер, 2005. – 736 с.
5. *Горбунов-Посадов М.М.* Расширяемые программы. – М.: Полиптих, 1999. – 336 с.
6. *Горбунов-Посадов М.М.* Как растет программа. – Препринт Института прикладной математики им. М.В. Келдыша РАН. – М., 2000 – 16 с.
7. *Холзнер С.* XSLT. Библиотека программиста. – СПб.: Питер, 2002. – 544 с.

**Колоколов Иван Анатольевич**
Федеральное государственное образовательное учреждение высшего профессионального образования "Южный федеральный университет" в г. Ростове-на-Дону.
E_mail: i__one@list.ru.
344015, г. Ростов-на-Дону, ул. Еременко, 60/2.
Тел.: 89044456227.

**Литвиненко Александр Николаевич**
E_mail: litva@rsu.ru.

**Kolokolov Inan Anatolevich**
Federal State-Owned Educational Establishment of Higher Vocational Education "Southern Federal University".
E_mail: i__one@list.ru.
60/2, Eremenko street, Rostov-on-Don, 344015, Russia.
Phone: 89044456227.

**Litvinenko Alexander Nikolaevich**
E_mail: litva@rsu.ru.