Метод управления разработкой пользовательского интерфейса для семейства программных продуктов

П.В.Коршунов
pav@tepkom.ru
Санкт-Петербургский государственный университет
198504, Университетский пр., 28
Санкт-Петербург, Россия

С.Джазабек
stan@comp.nus.edu.sg
National University of Singapore
117543, 3 Science Drive 2, Singapore

Аннотация

Работа посвящена проблеме управления процессом переиспользования пользовательского интерфейса при разработке семейства программных продуктов (product line). Рассматриваются графические интерфейсы, разрабатываемые в рамках модели MVC (Model-View-Controller). Для решения проблемы предлагается метод, который адаптирует XVCL-вариант фрейм-технологии, предоставляя схему и методику построения иерархии фреймов для графических интерфейсов. В работе представлено также программное средство, облегчающее практическое использования предлагаемого метода.

Введение

Успешная реализация программного продукта для конкретного заказчика (bespoke software)[11][1] влечет, как правило, появление новых заказчиков этого продукта с различными требованиями к нему. Таким образом, в результате возникают разные вер-

[1] В [11] рассматривается еще класс “коробочного” (shrink-wrapped) ПО.

сии продукта и может оказаться целесообразным создать *семейство программных продуктов*. Семейство программных продуктов (Product Line), согласно [8], является группой программных систем, обладающих общим набором свойств, удовлетворяющих нужды конкретного сегмента рынка либо имеющих сходное назначение, определенным образом разрабатываемых на основе общего набора ключевых активов.

В настоящее время большое внимание уделяется выработке методик и способов поддержки разработки и сопровождения семейства программных продуктов. Основной проблемой здесь является создание и использование общих активов семейства, в связи с чем развиваются специальные дисциплины и методы их переиспользования. Общими активами (core assets), согласно [8], являются документированные и сопровождаемые выделенные части ПО, которые могут быть компонентами кода, моделями предметной области, спецификациями и документацией, наборами тестов и т.д.

Данная работа использует XVCL[2]-технологию переиспользования[3] [4, 6, 14] артефактов процесса разработки, представленных в виде текстов. Технология основана на концепции архетипов и дельт, предложенной П. Бассетом [3]. XVCL-технология позволяет обеспечить переиспользование активов, выраженных в текстовом виде, и управлять отличиями конкретных реализаций в рамках семейства программных продуктов.

Разработка пользовательского интерфейса занимает, согласно [9], в среднем 45% ресурсов проекта, т.е. является трудоемкой частью процесса создания ПО. Пользовательский интерфейс имеет определенные особенности в реализации, связанные с его компонентной структурой и спецификой сочетания статического визуального представления и динамической обработки событий. Поэтому задачу управления его разработкой целесообразно рассматривать отдельно.

В статье представлен метод управления разработкой пользовательского интерфейса для семейства программных продуктов. В его рамках предлагается структурная модель пользовательского интерфейса, состоящая из нескольких уровней, в со-

[2]XML-based Variant Configuration Language.

[3]Данная технология разработана лабораторией программной инженерии (Software Engineering Lab) Национального университета Сингапура (National University of Singapore) под руководством С. Джазабека (S. Jarzabek).

ответствии с которыми должна строиться иерархическая схема фреймов. Также предлагаются правила организации кода пользовательского интерфейса во фреймы. Метод применим к интерфейсам, разрабатываемым в модели MVC [6].

В работе кратко описаны предлагаемые инструментальные средства, облегчающие создание фреймов и их иерархии, приводятся пример использования метода и краткий обзор существующих подходов переиспользования и средств создания пользовательского интерфейса.

1 Обзор работ в данной области

Существующие в настоящее время подходы ориентированы, главным образом, на создание пользовательского интерфейса, который в дальнейшем можно было бы переиспользовать. Имеется очень мало подходов, основанных на переиспользовании (возможно, уже созданного) пользовательского интерфейса в рамках семейства программных продуктов.

Далее рассмотрены основные существующие подходы и средства создания интерфейса, ориентированные на переиспользование.

1.1 Подходы к переиспользованию пользовательского интерфейса

Объектно-ориентированный подход (Object-Oriented Approach, OO-подход) основан на использовании полиморфизма методов классов и структурной организации классов с помощью отношения наследования. При этом код, относящийся к одному структурному элементу пользовательского интерфейса, помещают в один класс. Примерами объектно-ориентированных сред могут служить Swing, MFC и др. Более подробное описание OO-подхода и способов его применения можно найти в работе [5].

OO-подход достаточно естественен для создания пользовательского интерфейса так как его элементы (кнопки, текстовые поля, метки и т.д.) являются объектами с определенными состояниями и поведением, которые сравнительно легко описываются классами.

Основной проблемой при ОО-подходе является большое количество избыточного кода, сильное "дробление" классов, возрастание их количества при увеличении количества членов в семействе программных продуктов.

Аспектно-ориентированный подход, предложен Кижалешом [7] в 1997 г. Аспектом является часть функциональности и представления данных системы, которая не обособлена в ее архитектуре. Подход предоставляет концепцию, в которой проходящий сквозь всю систему аспект отделяется от остальной функциональности системы.

В контексте применения к переиспользованию пользовательского интерфейса в рамках семейства программных продуктов под аспектом можно понимать "сквозные" отличия интерфейса представителя семейства от переиспользуемого множества активов интерфейса [8].

Общей проблемой в AOP является четкое выделение кода аспекта из всего кода системы. К тому же при переиспользовании в рамках семейства далеко не все необходимые изменения, связанные с новым представителем семейства, являются аспектами.

Фрейм-технология переиспользования. Как уже было сказано, XVCL-технология, используемая в данной работе, основана на фрейм-технологии, которая, в свою очередь, строится на теории архетипов и родовой архитектуре, описанных в [3]. Технология фреймов уже на протяжении 20 лет применяется для создания переиспользуемого кода в семействах программных продуктов, написанных на языке COBOL. В работе П.Бассета [3] приводится количественная статистика переиспользования кода в COBOL-системах, полученная независимой организацией QSM. Утверждается, что процент переиспользования в результате применения данной технологии достигает 95%, цена проекта уменьшается на 84%, время разработки на 70%.

Основное "узкое место" данного подхода состоит в ориентируемости только на язык COBOL, что затрудняет использование в современных средах и языках программирования. Данное ограничение снято технологией XVCL.

1.2 Средства создания пользовательского интерфейса

Среди наиболее распространенных на сегодняшний день способов создания графического пользовательского интерфейса является его конструирование интерактивными средствами создания пользовательского интерфейса, поддерживающими RAD (Rapid Application Development). Большинство современных средств такого типа ориентировано на поддержку переиспользования. Другим альтернативным способом является генерация, средства которой позволяют не только достаточно быстро создавать интерфейс, но и достаточно эффективно решают задачу переиспользования.

Интерактивные графические средства как конструктор позволяют создавать пользовательский интерфейс системы из графических компонент. С помощью мышки можно, передвигая его элементы (widgets), создавать окна, диалоги системы, пользуясь такими понятиями, как визуальное размещение элементов (interface layout) и не заботясь о том, как выглядит код. Примерами таких средств являются VB, VC++ компании Microsoft, а также JBuilder компании Borland.

Все эти средства рассчитаны на использование объектно-ориентированного подхода (см. выше) при разработке пользовательского интерфейса.

Они позволяют относительно быстро строить интерфейс в основном небольших по размерам приложений. Они частично поддерживают создание переиспользуемых пользовательских интерфейсов, но необходимое внимание данной проблеме не уделено в должной мере, под переиспользованием понимается только локальное переиспользование, а переиспользование в рамках семейства программных продуктов не поддерживается. Также нет поддержки для управления разработкой в рамках семейства программных продуктов.

Средства генерации пользовательского интерфейса предназначаются для автоматизации процесса создания однотипных пользовательских интерфейсов. Это позволяет использовать средства генерации при создании интерфейсов различных членов семейства программных продуктов.

Генераторы являются программами, которые, имея на входе архитектурную модель системы (визуальные модели, описания на метаязыке), генерируют ее код. Использование генераторов позволяет абстрагироваться от низкоуровневых деталей реализации, что существенно упрощает процесс разработки и сопровождения системы, в том числе и пользовательского интерфейса.

Идея генерации по высокоуровневым моделям предлагает обещающее решение проблем создания, сопровождения, переиспользования кода. По сравнению с интерактивными средствами, тут уделяется внимание проблеме переиспользования в рамках семейства программных продуктов. Правда, на практике, в случае семейства, появляется необходимость в языке описания или моделирования, который будет поддерживать переиспользование ("семейственность"). Среди проблем, связанных с генераторами, можно также отметить сложность их создани и необходимость разработки нового генератора для новой архитектуры (таким образом зарождается уже семейство генераторов).

Генераторы могут быть успешно применены для переиспользования пользовательского интерфейса в некоторых определенных областях, например, в системах учета данных [2].

2 XVCL-технология переиспользования

В XVCL-технологии используется вариантно-конфигурационный препроцессорный язык XVCL [13]. Данный язык позволяет описывать переиспользуемые активы семейства программных продуктов и отличия между членами семейства в XML-формате.

Технология основана на понятии фрейма — произвольного текстового модуля, размеченного командами языка XVCL. С помощью данных команд фреймы могут изменять друг друга. Например, часть одного фрейма может быть объединена с частью другого, результат объединения сгенерирован в отдельный файл. Возможность изменения одного фрейма другим задает отношение адаптирования, на основе которого строится иерархия фреймов. Данное отношение задается командой "adapt" [13]. В текст фрейма, где описана команда "adapt", будет помещен результат препроцессирования того фрейма, имя которого указано

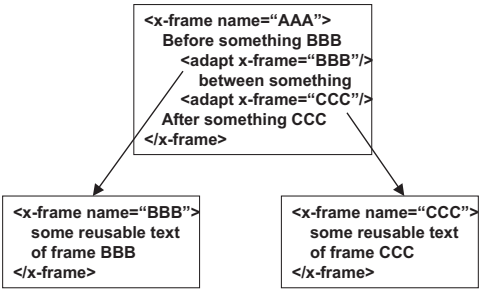

Рис. 1: Пример использования команды “adapt” языка XVCL

в качестве параметра этой команды (можно привести аналогию с макрокомандой “include” препроцессора языков C/C++). Иерархия фреймов, строго говоря, не является деревом потому, что один фрейм может быть адаптирован двумя и более фреймами.

На рисунке 1 фрейм с именем “AAA” (задается командой “x-frame” со значением атрибута “name”) содержит две команды “adapt” со значениями атрибута “x-frame” — “BBB” и “CCC” соответственно. Таким образом, фрейм “AAA” адаптирует фреймы “BBB” и “CCC”. Каждый из трех изображенных фреймов содержит текст, который может быть, например, кодом на языке Java. В результате запуска XVCL-препроцессора (см. ниже) будет сгенерирован текст:

```
Before something BBB
some reusable text of frame BBB
between something
some reusable text of frame CCC
After something CCC
```

Таким образом, вместо команды “adapt” включается результат препроцессирования соответствующего фрейма.

На основе команды “adapt” осуществляется переиспользование общих активов. Фреймы, не содержащие команды “adapt”, т.е. находящиеся в самом низу иерархии, являются в среднем более переиспользуемыми относительно фреймов, находящихся выше в иерархии. Корень иерархии фреймов называется спецификационным фреймом, или SPC-фреймом (Specification Frame). С его помощью задается иерархия фреймов для отдельного представителя семейства, включающая как переиспользо-

вание общих активов, так и описание его специфических черт [14]. Вообще говоря, допускается больше одного спецификационного фрейма в иерархии, таким образом, можно выделять определенные подграфы иерархии.

Команда “break” является командой разметки фрейма [13]. Команда помечает блок текста, который может быть удален/заменен/изменен с помощью других команд XVCL-языка.

Иерархия фреймов строится один раз для семейства программных продуктов. При создании в нее закладывается вариантность семейства и выделяются общие активы. Для каждого представителя семейства определяется отдельный спецификационный фрейм (их может быть несколько), в котором описывается, какие варианты в требованиях должны быть реализованы и какие активы переиспользованы. Другими словами, посредством спецификационных фреймов общая иерархия семейства “подгоняется” под представителя семейства.

Несомненным достоинством языка XVCL является возможность его применения для переиспользования различных составляющих программного продукта: кода системы, тестовых пакетов, технической и пользовательской документации, спецификации, требований, представленных в виде текста.

3 Метод разработки переиспользуемого пользовательского интерфейса

Предлагаемый метод представляет собой набор конкретных практических рекомендаций по организации пользовательского интерфейса в управляемую иерархию фреймов. Эти рекомендации основаны на определенной структуре пользовательского интерфейса, которая формализована посредством предлагаемой модели.

3.1 Модель пользовательского интерфейса

Визуальное представление интерфейса, на работу с которым ориентирован описываемый метод, можно выделить в системе, в которой используется модель пользовательского интерфейса MVC [10]. Она предполагает раздельную спецификацию бизнес-логики и визуального представления. В данном разделе предла-

Рис. 2: Модель пользовательского интерфейса

гается структурная модель[4] визуального представления интерфейса системы (рис. 2), которая определяет его четыре основных уровня:

- Уровень приложений — содержит части пользовательского интерфейса системы, не входящие ни в какую из программных компонент[5] продукта.
- Уровень компонент — включает части пользовательского интерфейса компоненты, которые не содержатся ни в какой определенной форме.
- Уровень экранов — состоит из набора форм, содержащих графические элементы пользовательского интерфейса.
- Элементарный уровень — содержит основные графические элементы пользовательского интерфейса, далее UI-объекты, такие, как кнопки, списки, метки и т.д.

На основе этой модели, используя XVCL-технологию, предлагается строить иерархию фреймов.

[4]Моделью является результат рассмотрения системы с определенной точки зрения [1].

[5]Под компонентой понимается функциональная компонента продукта, предоставляющая некоторую выделенную его функцию, как и определено в [11].

3.2 Способ построения иерархии фреймов для пользовательского интерфейса

Иерархия фреймов пользовательского интерфейса разделяется на подграфы, каждый из которых состоит из фреймов, соответствующих одному из уровней модели. Может быть несколько подграфов одного уровня. Каждый подграф характеризуется своим спецификационным фреймом (за исключением элементарного уровня модели). Так как в языке XVCL фрейм является файлом [13], то дополнительная структуризация иерархии может быть организована посредством каталогов файловой системы.

Иерархия фреймов. Код каждой формы пользовательского интерфейса помещается в отдельный фрейм. Если в новом продукте семейства нужна данная форма, то соответствующий ей фрейм присоединяется к иерархии фреймов пользовательского интерфейса семейства посредством команды “adapt”. Если форма состоит из элементов, большинство из которых соответствует различным вариантам семейства, то фрейм формы будет достаточно сложным. Тогда удобно создать отдельный фрейм, содержащий инициализацию формы, так как часто именно инициализация является большой выделенной частью кода формы (при использовании Java). Если определяется много XVCL-переменных во фрейме, то можно вынести их в отдельный фрейм опций.

Некоторые UI-объекты (или их набор) являются вариантами семейства — в этом случае их удобно поместить в отдельные фреймы. Тогда фрейм формы будет управлять или адаптировать все фреймы соответствующих элементов этой формы (т. е. визуального представления).

Спецификационный фрейм однозначно определяет подграф фреймов, который можно называть каркасом фреймов (framework). Фреймы форм пользовательского интерфейса соответствуют уровню экранов модели пользовательского интерфейса. Они адаптируют фреймы UI-элементов, составляющих формы, или элементарные. В небольших по размерам иерархиях фреймы, находящиеся над уровнем экранов, т.е. фреймы уровней приложений и компонент, объединяются в один уро-

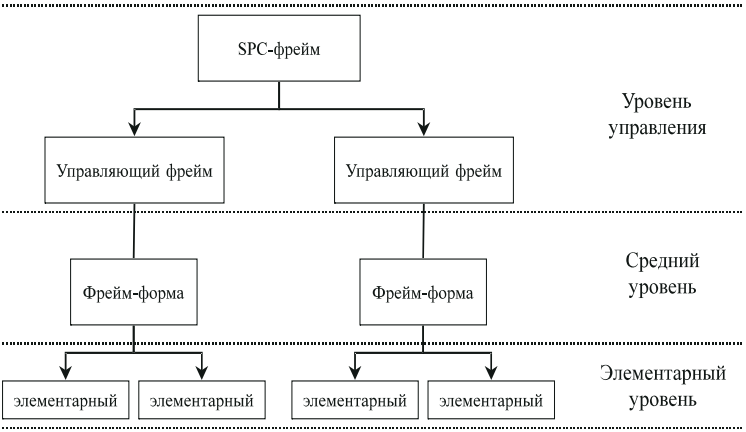

Рис. 3: Схема иерархии фреймов пользовательского интерфейса

вень управления (рис. 3). Во фреймах этого уровня определяются основные XVCL-переменные, соответствующие вариантам пользовательского интерфейса семейства. Фреймы управления, в свою очередь, адаптируются фреймами, координирующими между собой подграфы бизнес-логики (базы данных, алгоритмы) пользовательского интерфейса и, возможно, некоторых выделенных подсистем.

Структура фреймов элементарных объектов пользовательского интерфейса. Как уже было сказано, внизу иерархии находятся элементарные фреймы, соответствующие UI-объектам пользовательского интерфейса, т.е. текстовым полям, кнопкам, спискам и т.д. Нужно отметить, что UI-объект целесообразно вынести в отдельный фрейм, только если он является вариантом системы. При этом в каждый элементарный фрейм помещается код визуального представления соответствующего UI-объекта. Каждый элементарный фрейм, например фрейм кнопки, состоит из четырех основных частей. Он разделяется на части посредством команды языка XVCL “break”:

- Определение (definition). Определение элемента пользовательского интерфейса на исходном языке программирования.

- Инициализация (initialization). Все что касается инициализации этого элемента (размер, координаты, заголовок и т.д.).
- Действие (action). Тут находятся методы, реализующие обработчики событий по умолчанию. Например, для кнопки это может быть "actionPerformed()" (при использовании Java).
- Специальная (special). Эта часть обычно определяется пустой, на случай, если при адаптации понадобится вставить какой-нибудь дополнительный код.

При таком строении фрейма элементарного UI-объекта часть, относящаяся к представлению (определение и инициализация), отделена от части контроллера (действие). За счет гибкости языка XVCL, который позволяет генерировать результат препроцессирования разных фреймов в разные файлы, весь код, относящийся к одному UI-объекту, можно аккумулировать в одном фрейме, хотя в результате разные части кода могут оказаться в разных компонентах системы.

4 Инструментальные средства

На данный момент разработана инструментальная поддержка создания иерархии фреймов и ее проверки.

Редактор фреймов предназначен для создания и дальнейшей работе с фреймами. Основными его свойствами являются:

- поддержка команд языка XVCL;
- возможность просмотра структуры всей иерархии фреймов;
- возможность отображения структуры разметки каждого отдельного фрейма;
- интеграция с XVCL-препроцессором [12].

С помощью данного редактора можно:

- создавать и редактировать фреймы;
- осуществлять проверку на наличие ошибок в иерархии фреймов;

- осуществлять препроцессирование и генерировать результирующий код.

Редактор также отображает ошибки, возникающие в результате проверки фреймов на корректность, и реализует позиционное соответствие между структурой команд фрейма и его содержанием. Редактор интегрирован с *фреймовым препроцессором*, что позволяет проверить иерархию фреймов на синтаксическую и семантическую корректность и осуществить генерацию целевого кода из фреймов [12].

5 Пример использования метода

Метод построения иерархии фреймов, изложенный выше, был применен при создании переиспользуемого интерфейса небольшого исследовательского семейства CAD-систем[6]. Исходным языком программирования был язык Java. В результате была создана иерархия фреймов, задающая три разных варианта пользовательского интерфейса: автоматизированного места оператора, диспетчера и совмещенного места для оператора и диспетчера. За счет переиспользования общий размер кода уменьшился на 28%, при этом код фреймов содержит, помимо исходного языка, команды языка XVCL. Один из фрагментов иерархии семейства CAD-систем представлен на рисунке 4.

На рисунке виден спецификационный фрейм `CAD.spc`, в котором заданы изменения двух подграфов иерархии фреймов, относящихся к бизнес-логике и пользовательскому интерфейсу системы. Во фрейме `UICAD.spc` заданы варианты представления главных форм интерфейсов оператора и диспетчера. Он адаптирует инициализационную часть формы, заданной в `UIInit.xvcl` и фрейм `UIService.xvcl`, содержащий описание и обработку событий от элементов данной формы. Эти фреймы адаптируют элементарные фреймы, например, фрейм `UIInit.xvcl` добавляет в код инициализацию кнопок из `ButtonGroup.xvcl`, что представлено в приведенном ниже фрагменте:

[6]Computer Aided Dispatch Systems — программно-аппаратные системы, использующиеся в областях, где существует понятие инцидента (события), информация о котором сохраняется в базе данных и обработка которого осуществляется посредством диспетчеризации (при участии человека).

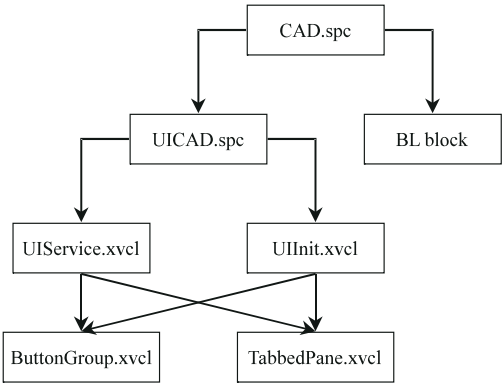

Рис. 4: Фрагмент иерархии фреймов пользовательского интерфейса CAD-системы

```
<set-multi var="loopNum" value="OK, Cancel, Apply"/>
<while using-items-in="?@loopNum?">
    <set var="name" value="CAD_button_?@loopNum?"/>
    <adapt x-frame="basicElements\ButtonGroup.xvcl">
        <insert break="definition"/>
        <insert break="action"/>
    </adapt>
</while>
```

В данном фрагменте определяется переменная языка XVCL — “loopNum”, которая затем используется в цикле и во фрейме `ButtonGroup.xvcl`. За счет использования “while” [13] будет сгенерирован Java-код инициализации (скрытый в `ButtonGroup.xvcl`) сразу для трех “кнопок”. В результате препроцессирования предложенного фрагмента Java-код определения и обработка событий “кнопок” не будут сгенерированы. Эти действия описываются аналогично в `UIService.xvcl` и могут быть, например, сгенерированы в другой файл.

Заключение

Предполагаются следующие направления дальнейшего развития представленного в работе метода:

- создание методики применения для промышленных проектов, специально отличая как небольшие (до 100 человеко-лет), так и крупные проекты (более 100 человеко-лет);
- поддержка при использовании метода циклической разработки программного обеспечения;
- расширение набора инструментальных средств поддержки метода;
- интеграция инструментальных средств поддержки метода с распространенными средами программирования, в первую очередь для языка Java: JBuilder, Idea и т.д.

Список литературы

[1] Кознов Д.В. Визуальное моделирование компонентного программного обеспечения: Дисс. канд. физ.-мат. наук. — СПб.: 2000. — 82 с.

[2] Стригун С.А. Опыт использования технологии REAL для создания информационных систем: Дипломная работа. — СПб.: 2001. — 56 с.

[3] Bassett P. Framing Software Reuse — lessons from the real world. — Prentice Hall: Yourdon Press, 1996. — 384 p.

[4] Cheong Y.C., Jarzabek S. Frame-based Method for Customizing Generic Software Architectures // Proc. of the Symposium on Software Reusability. — 1999. — P. 103-112.

[5] Fayad M., Schmidt D. Object-oriented Application Frameworks // Communications of the ACM. — Vol. 40. № 10. — 1997. — P. 32-38.

[6] Jarzabek S., Zhang H. XML-based Method and Tool for Handling Variant Requirements in Domain Models // Proc. of the 5th IEEE International Symposium on Requirements Engineering. — 2001. — P. 166-174.

[7] Kiczales G., Lamping J., Mendhekar A., Maeda C., Lopes C., Loingtier J-M., Irwin J. Aspect-Oriented Programming // Proc. European Conference for Object-Oriented Programming. — 1997. — P. 220-242.

[8] Linda M. Northrop SEI's Software Product Line Tenets // IEEE Software. — 2002. — Vol. 19. № 4. — P. 32-40.

[9] Myers B. A., Rosson M. B. Survey on user interface programming // Proc. of the Conference on Human Factors in Computing Systems. — 1992. — P. 192-202.

[10] Sandu D. User Interface Patterns // Proc. 8th Conf. Pattern Languages of Programs. — 2001.

[11] Sommerville I. Software Engineering (6th ed.). — Boston: Addison-Wesley, 2000. — 693 p.

[12] Wong T. W., Jarzabek S., Soe M. S. e.a. XML Implementation of Frame Processor // Proc. of the Symposium on Software Reusability. — 2001. — P. 18-20.

[13] XML based Variant Configuration Language — Specification, Version 2.0. — Singapore: 2002. — http://sourceforge.net/projects/fxvcl.

[14] Zhang H., Jarzabek S., Soe M.S. XVCL Approach to Separating Concerns in Product Family Assets // Generative and Component-based Software Engineering. — 2001. — P. 36-47.