$$3x_1 + 7x_2 + x_3 + 2x_4 + x_5 + 2x_6 + x_7 \le 15; \quad (7)$$

$$0 \le x_i \le 1, \ i = \overline{1,7}. \quad (8)$$

Решая эту задачу одним из известных методов, например, методом искусственного базиса [12], находим вектор, который показывает степень реализации функциональной полноты каждого СТ в рамках бюджета заказчика $\bar{x}(1; \frac{6}{7}; 1; 1; 1; 1; 0)$. Найденный вектор значений неизвестной переменной представляет собой конфигурацию СТ, удовлетворяющих заданным ограничениям. Можно выделить ряд факторов, которые оказывают влияние на полученный результат: опыт экспертов, оценка недоработки проекта и его функциональности, правильное и полное сопоставление параметров модели COCOMO II шаблону SRS, содержание объема работ для каждого СТ относительно полученного вектора результатов реализации функциональной полноты СТ. Данный подход дает возможность управлять СТ в рамках бюджета разрабатываемого проекта – нефункционального макро-требования заказчика.

Выводы. В работе поставлена и решена проблема расширения типовой функциональности СУТ возможностью поиска конфигураций требований с учетом ограничений. Научная новизна предложенного подхода состоит в совместном использовании модели COCOMO II и шаблонной спецификации на основе стандартных документов IEEE, а также представленной математической модели. Практическая значимость подхода состоит в повышении продуктивности аналитиков и менеджеров программных проектов в процессах разработки и управления системными требованиями. К недостаткам работы можно отнести то, что не предлагается способов автоматизации перехода от детальных спецификаций требований к параметрам модели COCOMO II. Для этого необходима особая формализация представлений требований, что является перспективой дальнейших исследований.

Список литературы: 1. *Соммервил И.* Инженерия программного обеспечения. – М.: Вильямс, 2002. – 624 с. **2.** *Бабенко Л.П., Лаврищева К.М.* Основи програмної інженерії. – К.: Знання, 2001. – 269 с. **3.** ISO 12207 Software Life-Cycle Processes. – 1995. – 57 с. **4.** IEEE/EIA 12207.1 – Industry Implementation of International Standard ISO/IEC 12207. – 1997. – 36 с. **5.** ISO/IEC 90003:2004 Software Standard**. –** 2004. – 175 p. **6.** IEEE Software Engineering Body of Knowledge Guide. 2004 – 200 с. **7.** Borland CaliberRM, www.borland.com/de/products/caliber. **8.** IBM Requisite Pro, www3.ibm.com/software/awdtools/reqpro. **9.** Telelogic Doors, www.telelogic.com. **10.** *Андон Ф.И., Коваль Г.И., Коротун Т.М.* Основы инженерии качества программных систем. – К.: Академпериодика, 2002. – 502 с. **11.** ISO 830-1998 Recommended Practice for Software Requirements Specifications. 1998. – 37 p. **12.** *Зайченко Ю.П.* Исследование операций. – К.: Вища шк., 1988. – 320 с.

Поступила в редакцию 10.10.2006

УДК 681.3

В.А. ШЕХОВЦОВ, канд. техн. наук, НТУ "ХПИ",
А.В. КОСТАНЯН, НТУ "ХПИ"

ОБРАБОТКА НЕФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ НА СТАДИЯХ АНАЛИЗА И ПРОЕКТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

У статті розглянуто підхід до аналізу нефункціональних вимог на перших стадіях розробки програмного забезпечення. Підхід базується на моделі KCPM, розширюючи її для обробки нефункціональних вимог до системи. Ці вимоги трактуються як сквозні інтереси. Представлені розширення моделі KCPM та нові правила відображення при використанні асиметричного AOSD підходу. Розглянуто також питання з використання методів NLP у даному підході.

This paper presents approach for non-functional requirements processing on early development stages. It's based on the Klagenfurt Conceptual Predesign Model (KCPM) allowing taking into account the non-functional requirements to the system. These requirements are treated as crosscutting concerns. The extensions of the KCPM schema and the new mapping rules are introduced for the case of asymmetric AOSD approach. Also questions about NLP support for our approach are considered.

Постановка проблемы. В процессе разработки программного обеспечения (ПО) всё более актуальной становится проблема обработки нефункциональных требований наряду с основными задачами, для решения которых данное ПО предназначается. Это обусловлено увеличивающейся сложностью создаваемых систем, временными ограничениями и повышением требований к самим программным продуктам, связанным со скоростью изменения окружающего мира. Под нефункциональными требованиями понимаются такие понятия как производительность, надёжность, эффективность, применимость, портируемость, тестируемость, воспринимаемость и модифицируемость программных систем. Проблема определения и обработки таких требований очень непроста. В работах [1, 2] были попытки описать некоторые подходы решения данного вопроса, но единой концепции не было. Положение изменилось с появлением АОРПО (Аспектно-Ориентированной Разработки Программного Обеспечения), которая выделяет нефункциональные требования как аспекты или сквозные интересы.

В данной статье мы опишем подход для выделения и конвертирования нефункциональных особенностей на этапе анализа требований в промежуточную модель, с последующим построением на её основе концептуальной модели системы. В качестве промежуточной модели мы используем расширенную KCPM (Klagenfurt Conceptual Predesign Model).

Анализ литературы. Опишем коротко основные положения и принципы используемых нами концепций. Таковыми являются АОРПО и KCPM. Дадим вначале определение АОРПО. Основной целью АОРПО является реализация средств систематической идентификации, разделения, представления и

компоновки сквозных интересов на различных этапах разработки программного обеспечения. Интерес (concern) представляет собой цель или концепцию, которая реализована в компоненте системы. Основным принципом в методологии АОРПО является разделение интересов (separation of concerns) [3]. В общем случае ПО рассматривается как реализация сложного взаимодействия различных базовых (core) и системных (system-level) интересов. Цели системных интересов – обеспечить взаимодействие между различными компонентами системы. Например, для системы обеспечения продаж набором основных интересов могут быть управление перечнями продукции и обработка заказов. Системные интересы в данном случае могут включать ведение журналов транзакций, обеспечение целостности и непротиворечивости данных, резервное копирование информации. Разделение интересов означает что различные виды или типы интересов должны быть идентифицированы и разграничены для данной задачи, чтобы понизить сложность разрабатываемой системы. Это разделение может проводиться на различных этапах проектирования ПО: анализе требований, проектировании и реализации. Но наиболее приемлемым решением является осуществление этого процесса на всех стадиях от анализа требований к реализации. Часто системные интересы затрагивают многие компоненты системы, принято говорить, что в такие интересы отражают сквозное поведение (crosscutting behavior) – поведение, которое присутствует во многих компонентах системы, возможно, не связанных друг с другом. Так, например, запись в системный журнал должна быть реализована в любом компоненте системы, поведение которого необходимо отслеживать. Интересы, связанные со сквозным поведением, еще называются сквозными интересами (crosscutting concerns) или аспектами (aspects). Сквозное поведение обладает рядом общих свойств:

1. Его реализация не является основной задачей компонентов системы.
2. Схема его организации не зависит от основной функциональности компонентов.
3. Такое поведение требуется во многих классах компонентов и во многих методах этих классов.

С реализацией сквозных интересов системы связан ряд проблем. Рассмотрим симптомы этих проблем:

1. Запутанность кода (code tangling). Если код метода содержит фрагменты реализации нескольких интересов, разобраться в организации такого кода становится сложно.
2. Разбросанность кода (code scattering). Невозможно выделить отдельный модуль, реализующий сквозной интерес – такая реализация распределена по многим модулям системы.

Запутанность и разбросанность кода приводят к следующим негативным последствиям для проектирования и разработки программных систем:

1. Снижение производительности разработки из-за того, что в ходе работы над методом класса внимание программиста все время перемещается с одного интереса на другой.
2. Ухудшение качества кода вследствие одновременного преследования нескольких целей при разработке метода. Нарушается принцип единственности реализации кода в системе, один раз исправленные ошибки могут возникать вновь.
3. Невозможность или сложность повторного использования кода, реализующего несколько интересов в условиях, когда необходима реализация только части из них.
4. Усложнение эволюции системы. При необходимости введения нового сквозного интереса (или переработки существующего) необходима модификация каждой подсистемы, в которой реализован этот интерес.

Выделяют два основных класса подходов к реализации систем на основе АОРПО: асимметричные и симметричные подходы. Основным принципом асимметричных подходов является то, что базовые и сквозные интересы выделяются отдельно и реализуются различным образом (obliviousness principle). При этом правила компоновки интересов задаются внутри реализации сквозных интересов (аспектов). Принцип отсутствия информации действует при разработке кода реализации базовых интересов. Наиболее известным подходом этого класса является аспектно-ориентированное программирование (АОП). Основным принципом симметричных подходов является то, что все интересы рассматриваются и реализуются одинаково. Правила компоновки интересов задаются отдельно от их реализации. Принцип отсутствия информации действует для всех интересов по отношению друг к другу. Наиболее известным подходом этого класса является многомерное разделение интересов (МРИ – MDSOC) [4]. АОП [5] реализует чёткое разделение между базовыми классами и аспектами. Базовые классы и аспекты сшиваются вместе в конечную систему. Рассмотрим основные конструкции АОП, которые мы будем использовать в дальнейшем:

1. Аспекты (aspects) – модули осуществляющие сквозное поведение. Каждый из них реализуется отдельно. На конечном этапе в полной системе производится композиция аспектов (aspect weaving).
2. Точки соединения (join points) – точно определённые места в базовых классах, такие как вызов метода или доступ к свойству. Модель точек соединения определяет множество позиций, в которых выполнение базовых классов может быть прервано для запуска сквозного кода.
3. Срезы (pointcuts) – правила, определяющие набор точек соединения. Они определяют места, где сквозной код перехватывает выполнение базовых классов. Например, вызов всех методов класса "Счёт".
4. Уведомление (Advice) – сквозной код аспекта, который запускается в точках соединения определённых срезом. Примером уведомления может служить запись журнала транзакций, запускаемая перед всеми вызовами

методов класса "Счёт". На стадии композиции, код уведомления внедряется в код базовых классов в местах, определённых срезом.

5. Интродукция – статическая часть уведомления. Интродукции представляют собой членов сквозного аспекта (методы и поля), которые внедряются в базовые классы в точках соединения определённых срезом. В качестве примера можно привести поле, содержащее имя файла журнала транзакций.

Одной из самых известных реализаций АОП является язык программирования AspectJ [6]. Это расширение языка Java, в котором определены конструкции для описания сквозного поведения. Традиционный аспектно-ориентированный процесс разработки ПО включает в себя на начальном этапе стадии анализа требований и концептуального моделирования или проектирования. При этом происходит резкий переход от первой стадии ко второй. Подразумевается, что данные и знания по предметной области легко могут быть извлечены из спецификаций требований и на их основе построена концептуальная модель системы. На практике же широко распространены случаи, когда такой подход достаточно сложно осуществим, и приводит к многочисленным ошибкам. Причиной является несоответствие концептуальной модели и спецификаций требований. Одним из подходов для решения данной проблемы является Klagenfurt Conceptual Predesign [7]. Основной целью этого подхода является реализация такой процедуры сбора требований, которая могла бы быть контролируема и проверяема заказчиком ПО. Данная цель достигается с помощью KCPM (Klagenfurt Conceptual Predesign Model) – промежуточной семантической модели, которая строится после стадии анализа требований перед построением концептуальной модели. Модель содержит семантические понятия более общие нежели при использовании концептуального моделирования. Они более понятны и доступны конечному пользователю (заказчику) и могут быть легко проверены им. В качестве таких понятий выступают:

1. Тип-сущность (thing-type) – обобщение понятий объекта, класса и атрибута.
2. Тип-связь (connection-type) – представляет собой все виды связей между элементами системы.
3. Операция (operation) – обобщение понятия метод.
4. Событие (event).

KCPM может быть представлена в табличной форме либо в виде семантической сети. В дальнейшем мы будем использовать табличное представление. После проверки пользователем модель может быть трансформирована в концептуальную модель. Это преобразование осуществляется с помощью правил преобразования (mapping rules). Мы рассмотрим глоссарии и правила конвертирования в следующих разделах.

KCPM может быть построена только на основании спецификации требований. При этом обычный подход позволяет использовать

исключительно функциональные требования. Построение может быть осуществлено двумя способами:

1. Произведено аналитиком.
2. Информация извлекается напрямую из текста спецификации с использованием Natural Language Processing (NLP).

Используя второй подход возможно построение концептуальной модели на основании текста спецификации. Этому посвящён проект NIBA [8], частью которого и является KCPM.

Цель статьи. Проанализировав текущую ситуацию, мы определили несколько проблематичных положений, требующих разрешения при осуществлении обработки нефункциональных требований на начальных этапах проектирования:

1. Построение промежуточной модели с использованием NLP методов. Возможности существующего подхода (NTMS парсер в проекте NIBA [9, 10]) весьма ограничены как в операционном плане (возможности по анализу текста) так и в области применимости (только немецкий язык). Это определяет необходимость создания более мощной и универсальной системы обработки текстов спецификаций.
2. KCPM на настоящий момент реализует трансформацию только для функциональных требований [11]. Это вызывает необходимость модификации модели для обработки нефункциональных требований. Только реализовав данную функциональность, можно будет получить законченную технологию для сбора и трансформации требований.
3. В процессе АОРПО отсутствует этап похожий по своей функциональной нагрузке на концептуальный предезайн (промежуточная модель). Соответственно возникает необходимость совместить преимущества АОРПО с возможностями, представляемыми ООП и KCPM.

Целью данной статьи является освещение первых шагов в создании технологий и подходов, позволяющих решить поставленные задачи.

Система анализа текстов спецификаций. Система для анализа спецификаций, представленных в виде текстов на естественном языке, должна включать в себя следующие элементы:

1. Лингвистический словарь.
2. Словарь понятий, представляющий собой значения слов.
3. Синтаксический анализатор (грамматика).
4. Интерпретатор, позволяющий корректно извлечь необходимую информацию на основании значений слов и грамматических конструкций в зависимости от контекста.

Одним из широко известных проектов, в рамках которого реализована система со всеми перечисленными выше элементами, является NIBA. На его основании возможно построение интересующей нас системы анализа спецификаций при изменении следующих особенностей реализации:

1. Предложения разбираются в древовидную структуру, в вершине которой находится глагол. При этом существует определённый набор "классов" глаголов, для которых имеется определённая структура предложения. Синтаксический анализатор (парсер) создан с использованием Пролога и Definite Clause Grammar (DCG) [10]. Это говорит о "зашитости" возможных структур предложения и тяжёлой модификации этих правил в дальнейшем. Отсюда возникает необходимость разработки открытой архитектуры для задания правил разбора предложений и синтаксических конструкций.

2. На вход парсера подаётся предложение только на немецком языке.

Это говорит о том что в структура парсера рассчитана исключительно на этот язык и его особенности. Возникает необходимость создания парсера, способного к распознаванию набора языков (минимум европейской группы). Эта ситуация связана с созданием новой (более гибкой и модифицируемой) структуры для словарей.

Следует также учитывать, что для парсеров с использованием DCG и Пролога характерны следующие проблемы:

1. Лингвистическая грамматика имеет очень большой объём.

2. Высокая степень неопределённости (неоднозначности) данных на выходе парсера.

3. Необходимость иметь большой словарь с высокой скоростью доступа к данным.

Все вышеперечисленные проблемы вызывают необходимость обратиться к иным лингвистическим подходам, позволяющим анализировать текст на естественном языке. Одним из наиболее подходящих из них для решения нашей задачи является так называемая "Аффиксная Грамматика для Конечных Решёток" (Affix Grammar over a Finite Lattice – AGFL [13]). В ней решены проблемы, характерные для DCG. Кроме того, её использование позволяет создавать парсеры текстов на различных языках, используя унифицированный механизм интерпретации синтаксических конструкций [13 – 16]. Именно данный подход мы выбрали для основы системы анализа текстов спецификаций.

Аспектный предезайн. Рассмотрим вопрос представления и преобразования нефункциональных правил используя KCPM. Для обработки этих требований будем следовать АОРПО подходу, который определяет их как сквозные интересы (аспекты) системы. Для обозначения происхождения требований, с которыми мы будем работать, мы назвали нашу расширенную технику предезайна "аспектный предезайн". Существует два различных этапа, выполняемых в процессе предезайна [7]:

1. Заполнение обобщённой KCPM схемы требованиями к системе с помощью пользователя (заказчика).

2. Осуществление преобразования данных из KCPM схемы в концептуальную модель данных.

Для обработки нефункциональных требований оба этапа необходимо подвергнуть изменениям. Основной проблемой аспектного предезайна является необходимость определения семантических понятий, которые бы адекватно представляли сквозные интересы в системе. При этом понятия должны оставаться легко доступными конечному пользователю, который осуществляет проверку данных. Обеспечить адекватность и понятность одновременно достаточно непросто. Главной причиной является терминология АОРПО. Особенно это заметно при использовании АОП, при использовании таких понятий как аспекты, точки соединения, срезы, уведомления, интродукции и т.д. Данная терминология совершенно непонятна конечному пользователю, потому как:

1. АОП было создано профессиональными программистами, проблемы которых отличаются от проблем обычных пользователей.

2. Полную технологию сквозных интересов достаточно сложно освоить даже опытным разработчикам и архитекторам ПО.

Проблема сложной терминологии распространяется не только на конечных пользователей, но и на аналитиков. Например, для того чтобы промоделировать точки соединения с помощью UML [12], необходимо изменить их определение по нотации AspectJ с "принципиальные точки выполнения программы" [6] на более общее – "места для добавления расширений". В данной статье мы рассмотрим первые шаги для решения описанной проблемы. Наш подход упрощён, так как мы не рассматриваем сложные определения срезов, однако может быть с успехом применён для более сложных случаев. Для дальнейшего описания подхода нам необходимо определить предметную область и привести пример спецификации требований, на основании которого мы рассмотрим наш подход. Рассмотрим систему для работы в банке. Фрагмент описания требований выглядит следующим образом: "Банковская система работает с клиентами и счетами. Все операции со счетами необходимо сохранять (протоколировать) в файл. При попытке клиентов снять средства со счёта, они все обязательно должны ввести в начале пароль". Для моделирования мы будем использовать KCPM в табличном представлении. Рассмотрим моделирование интересов. В начале определим, что семантическим понятием для интереса является тип-сущность. Для того, чтобы различать аспектные тип-сущности от стандартных, классификационная колонка для них будет содержать значение "интерес" (concern). Проектировщик ответственен за установку данного значения. Тип-сущности с учётом аспектов для нашей предметной области представлены в табл. 1. Здесь в качестве аспектов определены "безопасность" и "протоколирование".

Рассмотрим моделирование уведомлений. Уведомление достаточно схоже с операцией, однако отличается от неё механизмом вызова. Оно вызывается неявно в точках соединения, связанных с ним посредством среза. В нашей модели мы утверждаем следующее:

1. Тип-сущность на которое ссылается уведомление должен быть аспектом.

2. Колонки для описания типа операций должны содержать значения "авто-вызов".

Таблица 1

Фрагмент глоссария тип-сущностей с моделированием интересов

Предметная область: банковская система				
Код#	Название	Классификация	…	Источник в специф.
*D*001	Безопасность	интерес	…	*S*3
*D*002	Протоколирование	интерес	…	*S*2
*D*003	Счёт	тип-сущность	…	*S*1, *S*2, *S*3
*D*004	Клиент	тип-сущность	…	*S*1, *S*3
…	…	…	…	…

Для обеспечения возможности вызова уведомлений необходима дополнительная информация, обеспечиваемая срезами, которая будет описана ниже. Фрагмент глоссария для операций с учётом уведомлений представлен в табл. 2.

Таблица 2

Фрагмент глоссария операций с моделированием уведомлений

Предметная область: банковская система				
Код#	Название	Тип вызова	…	Исполняющая тип-сущность
*O*001	Снятие средств	явный	…	*D*004
*O*002	Запрос пароля	авто-вызов	…	*D*001
*O*003	Протоколирование	авто-вызов	…	*D*002

Рассмотрим моделирование интродукций. В технологии АОП существует два типа интродукций: методов и полей. Их семантическими соответствиями являются операции и тип-сущности. Единственное различие между интродукцией методов и уведомлением является тип вызова, который в случае интродукции должен содержать значение "авто-вставка". Тип-сущности, представляющие интродукции полей должны быть связаны с тип-сущностями интересов. Для обеспечения неявного применения интродукций также необходима информация, предоставляемая срезами. Эта информация носит тот же характер, что и для уведомлений.

Моделирование точек соединения. На практике большинство точек соединения в спецификациях требований могут быть разбиты на две категории:

1. Вызов какой-либо операции. В терминах АОП вызов какого-либо метода.

2. Доступ к какой-либо тип-сущности.

Соответствующие категории в АОП, применяемые на следующих этапах разработки, могут так же иметь две различные реализации:

1. Доступ к соответствующему полю или классу.

2. Вызов всех методов какого-либо класса.

Таким образом, можно утверждать следующее: в большинстве случаев точки соединения могут быть представлены либо операцией либо тип-сущностью. Ниже мы опишем способ применения такого представления.

Моделирование срезов. Срез представляет собой связь между уведомлением и точкой соединения. Таким образом, он может быть определён как тип-связь. Чтобы позволить типу-связи представлять срез, необходимо изменить структуру записи для него. Нам необходимо обеспечить возможность ссылки среза не только на тип-сущность, но и на уведомления, с которыми он связан. Для этого необходимо разделить колонку "связанный тип-сущность" на две:

1. "связанный тип-код" – будет содержать код ссылаемого типа.

2. "связанный тип" – будет содержать тип связанного понятия (операция или тип-сущность).

Больше никаких изменений не требуется. В табл. 3 представлен фрагмент глоссария тип-связей для выбранной нами предметной области и её спецификации. Срез *C*001 соединяет уведомление для запроса пароля с вызовом операции снятия средств. Срез *C*002 соединяет уведомление протоколирования со всеми операциями тип-сущности "Клиент".

Таблица 3

Фрагмент глоссария тип-связей с моделированием срезов

Предметная область: банковская система							
			Проекция				Источ-ник
Код#	Название	…	Код проекции	Связанный тип-код	Связанный тип	Название	
*C*001	Пароль для операции	…	*P*001a	*O*002	операция	Запрос пароля	*S*3
			*P*001b	*O*001	операция	Снятие средств	
*C*002	Протоко-лирова-ние всех операций	…	*P*002a	*O*003	операция	Протоколи-рование	*S*2
			*P*002b	*D*003	тип-сущность	Счёт	
…	…	…	…	…	…	…	…

Расширение процесса преобразования. Прежде всего, для описания процесса преобразования необходимо выбрать конечную модель либо нотацию. Так как КСРМ использует UML (Unified Modeling Language), то и мы остановимся на ней. В последнее время было проведено множество исследований для адаптации UML к описанию сквозного поведения системы. В результате появилось официальное расширение UML для АОРПО. Однако мы воспользуемся более специфической нотацией, в основе которой лежит использование стереотипов – Aspect-Oriented Design Model (AODM) [12]. Она содержит понятия для представления аспектов, уведомлений, срезов, точек соединения интродукций и является более наглядной конечной моделью. Преобразование включает в себя выполнение следующих десяти аспектных правил:

1. Правило аспекта. Тип-сущность T переходит в аспект At в случае пометки T проектировщиком как "интерес" в поле "Классификация".

После выполнения данного правила все операции, связанные с данным типом-сущностью, будут преобразованы в уведомления либо в интродукции методов в соответствии с приведёнными ниже правилами. Все тип-сущности, связанные с данной, будут преобразованы в интродукции полей.

2. Правило уведомления. Операция O переходит в уведомление ADo, если O ссылается на тип-сущность, преобразованный в аспект, и O помечена как "авто-вызов" в поле "тип вызова".

3. Правило интродукции метода. Операция O переходит в интродукцию метода IMo, если O ссылается на тип-сущность, преобразованный в аспект, и O помечена как "авто-вставка" в поле "тип вызова".

4. Правило интродукции поля. Тип-сущность T переходит в интродукцию поля IFT, если T связан с другим типом-сущностью, который уже преобразован в аспект.

5. Правило среза № 1. Тип-связь C переходит в срез PC, если C ссылается на операцию, уже преобразованную в уведомление.

6. Правило среза № 2. Тип-связь C переходит в срез PC, если C ссылается на операцию, уже преобразованную в интродукцию метода.

7. Правило среза № 3. Тип-связь C переходит в срез PC, если C связана с типом-сущностью, уже преобразованном в интродукцию поля.

После применения любого правила среза вторая часть связи будет преобразована в точку соединения с помощью одного из правил точек соединения.

8. Правило точки соединения № 1. Операция O переходит в точку соединения JO, если O уже преобразовано в метод Mo и связана с типом-связью, уже преобразованным в срез. В этом случае JO представляет собой вызов метода Mo.

9. Правило точки соединения № 2. Тип-сущность T переходит в точку соединения JT, если T уже преобразован в тип-значение VT и связан с типом-

связью, уже преобразованным в срез. В этом случае JT представляет собой доступ к этому типу-значению (атрибуту).

10. Правило точки соединения № 3. Тип-сущность T переходит в точку соединения JT, если T уже преобразован в класс C и связан с типом-связью, уже преобразованным в срез. В этом случае JT представляет собой вызов любого метода класса C.

Для преобразования модели из нашего примера необходимо выполнить следующие аспектные правила:

1. Правило аспекта для преобразования тип-сущностей "Безопасности" (*D*001) и "Протоколирования" (*D*002) в аспекты.

2. Правило уведомления для преобразования операций "Запрос пароля" (*O*002) и "Протоколирование" (*O*003) в уведомления.

3. Правило среза для преобразования тип-связей *C*001 и *C*002 в срезы.

4. Правило точек соединения для преобразования операции "Снятие средств" (*O*001) и тип-сущности "Счёт" (*D*003) в точки соединения.

Приведенная упрощённая спецификация требований не требует применения правил для преобразования интродукций.

Выводы. В данной статье описаны первые шаги в решении проблемы извлечения и обработки нефункциональных требований на начальных стадиях разработки программного обеспечения. Для проверки применимости наших идей необходимы реальные проекты и пользователи. Только после проверки в реальных условиях и, возможно, произведения изменений как в модели, так и в правилах преобразования можно будет утверждать о законченности нашего подхода. Открытым так же является вопрос с созданием автоматизированной системы анализа спецификаций требований, представленных в виде текстов. В этом направлении мы предполагаем разработку системы, аналогичной проекту NIBA, но при этом лишённой его недостатков.

Список литературы: 1. *Chung L., Nixon B., Yu E., Mylopoulos J.* Non-Functional Requirements in Software Engineering. Boston: Kluwer Academic Publishers, 2000. **2.** *Mylopoulos J., Chung L., Nixon B.* Representing and Using Non-Functional Requirements:A Process-Oriented Approach. In: IEEE Transactions on Software Engineering. – Vol. 18. – №. 6. – P. 483–497. **3.** *Dijkstra E.* Discipline of Programming. Prentice-Hall, 1976. **4.** *Tarr P.; Ossher H., Harrison W., Sutton S.M.* N Degrees of Separation: Multi-Dimensional Separation of Concerns. Proc. ICSE'99 Intl. Conference, May, 1999. **5.** *Kiczales G., Lamping J., Mendhekar A.* Aspect-Oriented Programming. In: Proceedings of the European Conf. on Object-Oriented Programming (ECOOP), 1997. **6.** AspectJ Team: The AspectJ Programming Guide, 2004. URL: http://aspectj.org/doc/dist/progguide/index.html. **7.** *Mayr H.C., Kop Ch.* A User Centered Approach to Requirements Modeling. In: M.Glinz, G. Müller-Luschnat (eds.): Proc. Modellierung 2002. Lecture Notes in Informatics P-12 (LNI), GI-Edition. – 2002. – P.75–86. **8.** *Niba L.C.* The NIBA workflow: From textual requirements specifications to UML schemata. In: Proc. ICSSEA'2002, Paris. – 2002. **9.** *Mayerthaler W., Fliedl G., Winkler Ch.* Lexikon der Natürlichkeitstheoretischen Syntax und Morphosyntax, Stauffenburg Verlag, Brigitte Narr, Tübingen. – 1998. **10.** *Fliedl G.* Naturlichkeitscheoretische Morphsyntax, Aspekte der Theorie und Implementierung. Habilitationsschrift, Gunter Narr – Verlag. – 1999. **11.** *Kop Ch., Mayr H.C.* Mapping Functional Requirements: From Natural Language to Conceptual Schemata. In: Proc. International Conference SEA 2002. – Cambridge, USA. – P. 82–87. **12.** *Stein D., Hanenberg S., Unland R.* Modeling Pointcuts. In: Early Aspects 2004: Workshop at International Conference on Aspect-Oriented Software Development

(AOSD 2004), Lancaster, 2004. – P. 107–113. **13.** *Koster C.H.A.* Affix grammar for Natural Languages. In: H. Alblas and B. Melichar (Eds.), Attribute grammars, applications and systems.SLNCS 45, Heidelberg, 1991. – P. 469–484. **14.** *Farkas E., Koster C.H.A.* Koves and M. Naszodi. Towards an Affix Grammar for the Hungarian language. In: A. Gyorgy (Ed.), Proceedings CIS'91 conference on intelligent Systems, John von Neumann Society for Computing Sciences, Budapest 1991. – P. 233–246. **15.** *Koster C.H.A., Willems R.* Towards an Affix Grammar for Turkish. In: M. Baray and B. Ozguc (Eds.), Computer and Information Science VI, Elsevier, Amsterdam 1991. – P. 1067–1076. **16**. *Maria Paula Santalla del Rio A* Formal Grammar of Spanish for Phrase-Level Analysis Applied to Information Retrieval, Lalia, 14, Series Maior, University of Santiago de Compostela, Santiago de Compostela, 2002.

Поступила в редакцию 23.10.2006

Содержание