Визуальные языки проектов

Д.В.Кознов* Л.Б.Ольхович†
dim@tepkom.ru leo@tepkom.ru

Санкт-Петербургский государственный университет
198504, Университетский пр., 28
Санкт-Петербург, Россия

Аннотация

Данная работа посвящена исследованию визуальных языков проектов — диалектов стандартных языков (UML, SDL и т.д.), адаптирующих эти языки к особенностям использующих их проектов. Рассматривается процесс возникновения этих языков, вопросы их формализации и переиспользования. Визуальные языки проектов рассматриваются в контексте следующих способов использования визуального моделирования: а) как средства коммуникации (в частности, при анализе и проектировании); б) с применением автоматической обработки (генерации кода, тестов и т.д.). Приводятся фрагменты визуальных языков проектов, созданных в рамках промышленных проектов. Рассматривается основанный на использовании языка OCL метод автоматизированной поддержки синтаксической корректности использования визуального языка проекта.

Введение

Как показывает практика, визуальное моделирование, подобно другим активам программной инженерии, при использовании предполагает дополнительную работу по адаптации. В [2] показано, что эффективная автоматическая обработка визуальных спецификаций (генерация кода, тестов, поиск "мертвых" циклов в алгоритмах, оценка непротиворечивости моделей и пр.)

*Работа поддержана грантом PD02-2.8-145 Министерства образования России.
†Работа поддержана грантом Министерства образования России по исследованиям в области гуманитарных, естественных, технических и медицинских наук.

требует, как правило, создания специальных, ориентированных на проект решений (технологических решений) — набора стандартных средств визуального моделирования (языков, методов, CASE-пакетов), настроенных на решение проблем данного проекта, а также специально созданных программных средств, дополняющих функциональность стандартных.

Однако даже при отсутствии автоматической обработки визуальных спецификаций, ограничиваясь использованием визуального моделирования лишь как средства коммуникации участников (stakeholders) программного проекта (в частности, при анализе и проектировании), все равно встает проблема адаптации стандартных языков и методов их использования к потребностям конкретных проектов.

В данной работе, основываясь на семиотическом подходе[1], исследуются особенности применения визуального моделирования в конкретных проектах.

В работе вводится понятие визуального языка проекта — диалекта стандартного языка (UML, SDL и т.д.) или группы таких языков, адаптирующего эти языки к использующему их проекту.

Рассматривается процесс возникновения таких языков, а также вопросы их формализации и переиспользования. Выделяются два специальных способа применения визуального моделирования: как средства коммуникации (в частности, при анализе и проектировании), а также в контексте автоматической обработки. Рассматривается основанный на использовании языка OCL метод автоматизированной поддержки синтаксической корректности спецификаций, выполненных с помощью визуального языка проекта. Приводятся примеры визуальных языков проектов из промышленных разработок компании ЗАО ЛАНИТ-ТЕРКОМ.

[1]Семиотика, согласно Чарльзу Моррису, является наукой, изучающей вещи и свойства вещей в их функции служить знаками [4]. Эта наука, таким образом, делает попытку построить общую теорию знаковых систем, применимую в различных областях — лингвистике, логике, философии, психологии, биологии, антропологии, программировании и т.д.

1 Визуальное моделирование

Будем называть визуальным моделированием регулярное использование изображений, основанных на графах, для последовательного построения моделей существующих и создаваемых программных систем. На эти модели наложим дополнительные ограничения — они должны создаваться из небольшого набора стандартных заготовок в соответствии с определенными правилами, т.е. на основе специальных языков.

Использование визуального моделирования в программировании восходит к работам фон Неймана, который предложил язык блок-схем в качестве средства спецификации математических алгоритмов для их переноса на вычислительные машины [18]. Впоследствии для блок-схем было создано большое количество различных стандартов как в Европе, так и в Америке. Можно сказать, что работы фон Неймана породили две ветви в области средств визуализации в программировании — визуальное программирование и визуальное моделирование. Второй подход, в отличие от первого, не ставит своей задачей разработку средств визуализации всего приложения, останавливаясь лишь на отдельных, значимых для проекта аспектах. Визуальное программирование (обзор подходов можно найти, например, в [23]) на настоящий момент не продвинулось дальше экспериментальных разработок.

В конце 1960-х гг. визуальное моделирование было положено в основу метода SADT[2], активно использующегося до настоящего времени. В 1970-х гг. был создан язык SDL[3] для спецификации телекоммуникационных систем, который продолжает использоваться и развиваться и в настоящее время. В 1970-1980-е гг. активно развивается структурный анализ. В 1980-1990-е гг. визуальное моделирование широко используется в рамках CASE[4]-средств для разработки информационных систем — такие системы как ADW, CoolGen, SEER*HPS[5] имели многомил-

[2]Structured Analysis and Design Technique — метод анализа и проектирования различных искусственных систем [19].

[3]Specification and Description Language — стандарт международной организации ITU (International Telecommunication Union) [25].

[4]Computer Aided Software Engineering — разработка ПО с использованием компьютерных средств автоматизации.

[5]ADW (Application Developing Workbench), CoolGen и SEER*HPS — CASE-системы, предназначавшиеся для корпоративной разработки и сопровождения

лионные продажи. В 1990-х гг. бурно развивается объектно-ориентированный анализ и проектирование, что в 1997 г. привело к созданию унифицированного языка визуального моделирования — UML[6].

Долгое время развитие визуального моделирования рассматривалось как один из основных способов превращения программирования из искусства в строгую инженерную дисциплину [24, 17]. Однако со временем выявились следующие трудности в его использовании: растет количество дополнительной работы в проекте, увеличивается число элементов конфигурационного управления, затруднен переход от диаграмм к программному коду. Средства автоматизации визуального моделирования (CASE-пакеты) являются одними из самых невостребованных после их приобретения программными продуктами [7].

2 О вариативности знаковых систем, используемых в программировании

2.1 Определения

Согласно [4], основными измерениями знаковой системы являются: а) синтаксис — отношение знаков между собой; б) семантика — значения знаков, то есть их отношение с объектами внешнего мира; в) прагматика — отношение знаков и их интерпретатора (т.е. пользователя знаковой системы).

Знаковые системы, используемые при разработке ПО, имеют:

- стандартную часть, переиспользуемую от одной конкретной разработки к другой, которую мы будем называть стандартным языком;
- вариативную часть, позволяющую учитывать нестандартные особенности конкретного проекта.

Итоговую знаковую систему, используемую в программном проекте, будем называть языком проекта. Мы будем рассматривать

многоплатформенных информационных систем.

[6]Unified Modeling Language — стандарт международной организации OMG (Object Management Group), предназначенный для объектно-ориентированного визуального моделирования ПО [21]. В данный момент является основным языком визуального моделирования, используемым в промышленном программировании.

визуальные языки проекта, а также остановимся на вариативности языков программирования для того, чтобы понять причину высокого уровня изменчивости стандартных визуальных языков.

Для удобства построения дальнейших рассуждений в начале дадим определения синтаксиса, семантики и прагматики для языка программирования и визуального языка.

Синтаксисом языка программирования будем считать правила построения соответствующих ему текстов, т.е. его грамматику. Будем считать, что множество объектов, на которые указывают его конструкции, задается абстрактным вычислительным процессом. Именно в терминах таких вычислительных моделей обычно определяется семантика языка программирования (так называемая "операционная семантика"). Его прагматикой будем называть способы применения этого языка программистами[7]. Основной фабулой эволюции языков программирования является баланс семантики и прагматики — эффективных вычислительных возможностей языка и удобств его использования.

Синтаксисом визуального языка будем считать правила, регламентирующие структуру его диаграмм. Определим, что источник объектов, на которые указывают конструкции визуального языка, задается моделируемой системой и артефактами процесса ее разработки. Прагматику определим как различные методы использования визуального языка.

2.2 Вариативность языков программирования

В теории языков программирования известны понятия над- и подъязыков: "если определяется язык A, собственно-программы которого являются также собственно-программами языка B, и смысл собственно-программы, определяемой языком A, совпадает с ее смыслом, определяемым языком B, то A называется «подъязыком» для B, а B называется «надъязыком» для A" [1].

При создании языка проекта вместо понятий надъязыка и подъязыка удобно использовать понятия операций *расширения* и *усечения*. В этом случае язык проекта является результатом

[7]Мы не пользуемся здесь стандартными определениями синтаксиса, семантики и прагматики языков программирования (см., например, [1]) так как нам нужны определения этих понятий, удобные как для визуальных языков, так и для языков программирования.

выполнения последовательности таких операций над стандартным языком, не являясь, строго говоря, ни его надъязыком, ни подъязыком.

Многие ранние языки программирования — COBOL, PL/1 и др., фактически, имели столько вариантов, сколько существовало для них компиляторов. Впоследствии для промышленных языков программирования стали создаваться стандарты — ANSI C/C++, SQL и т.д. С другой стороны, рынок компиляторов существенно сократился.

При применении языков программирования порождаются вариации: некоторые конструкции могут полностью или частично не использоваться (что означает усечение синтаксиса), в то же время могут создаваться новые (расширение синтаксиса) с помощью специальных механизмов, встроенных в языки, в частности, через создание макроопределений и библиотек. Изменения семантики ограничены требованием сохранения операционной семантики языка, что связано с практикой использования стандартных компиляторов и сред разработки, которые должны "понимать" язык проекта.

2.3 Вариативность визуальных языков

Попытки сделать отправной точкой семантики визуальных языков операционную семантику, как в случае с языками программирования, дают лишь частичные результаты — можно назвать язык IDEF1x[8], используемый для разработки и автоматической генерации схемы базы данных, и язык SDL, успешно используемый при разработке телекоммуникационных алгоритмов с последующей автоматической генерацией целевого кода. В общем случае предназначение визуальных языков иное, чем языков программирования — они используются для коммуникации различных специалистов программной разработки, а не для взаимодействия человека и машины.

Попытка создать хорошо определенную неоперационную семантику для визуального языка реализована в рамках подхода SADT [19]. Эта попытка оказалась успешной ввиду создания де-

[8]Integration Definition For Information Modeling — графический язык и методика его использования, созданные для моделирования данных для информационных систем. Является федеральным стандартом США.

тально проработанной прагматики[9] и ограничения области применения языка — структурный анализ искусственных систем.

В UML сделана попытка создать абстрактную универсальную семантику, ориентированную на моделирование ПО. Однако эта семантика оказывается очень громоздкой и трудной в использовании. На практике уточнение семантики синтаксических конструкций в рамках конкретного проекта происходит не в соответствии с этой абстрактной семантикой, а во многом "ad hoc".

Семантику визуального языка (т.е. его связь с обозначаемыми объектами) не удается столь строго определить, как в случае с языками программирования — слишком велико многообразие создаваемых систем и не существует стандартного процесса их разработки.

Язык UML обладает наибольшей вариативностью среди современных визуальных языков: при его использовании применяются не только операции расширения и усечения, но и изменения. Кроме того, поскольку в нем никак не фиксируется прагматика[10], то в каждом конкретном случае ее нужно определять.

Такие свободы в области визуального моделирования сопровождаются необходимостью доработки стандартных средств автоматизации, особенно при автоматической обработке визуальных спецификаций. Эта работа гораздо более трудоемка, чем настройка стандартного компилятора. Именно поэтому оказывается востребованным понятие технологического решения [2].

3 Обзор подходов к учету вариативности визуальных языков

Существует большое количество технических подходов к адаптации визуального моделирования. Настраиваемые CASE-пакеты (см., например, [15]), которые ориентированы на широкую область применения, позволяют изменять широкий спектр параметров. Meta-CASE-средства [20] предназначены для генерации CASE-пакетов, поддерживающих любой специфицирован-

[9] То есть метода.

[10] UML не включает в себя метод разработки ПО, что отличает его от многочисленных работ начала/середины 1990-х гг., предлагавших различные подходы к объектно-ориентированному анализу и проектированию ПО, в которых язык визуального моделирования рассматривался как приложение к методу.

ный язык визуального моделирования. Однако эти и подобные подходы не исследуют контекст модификации стандартных средств визуального моделирования как целостное явление (т.е. его причины, эволюционные и психологические аспекты и т.д.), в силу чего их практическое применение затруднено.

Концепция профилей UML, предполагающая создание вариаций UML для разных предметных областей, очень близка к понятию визуального языка проекта. Однако профили не предназначены для модификаций языка UML с учетом специфики проекта. По сути, каждый профиль — это новый стандарт, преследующий своей целью унификацию изменений, вносимых в UML при его использовании в рамках некоторой области. В то время как визуальные языки проектов появляются спонтанно и эволюционируют по мере развития проекта, профили создаются осознанно, и, как любые стандарты, не предполагают частых изменений.

Многие проекты обладают уникальными особеностями, не учтенными в существующих профилях. Следствием этого является потребность в методике построения частных решений, удовлетворяющих локальным нуждам проекта. Предлагаемый в работе подход — формализация и спецификация визуального языка проекта — является продолжением идеи формализации других информационных активов программного проекта: словаря предметной области [26], неформальных знаний, содержащихся в текстах программ в контексте задачи автоматического реинжиниринга [13], стилей программирования в проекте и т.д.

4 Визуальное моделирование как средство коммуникации

В [11] при обсуждении метода случаев использования (use case approach) приводится несколько точек зрения на то, как этот метод применять: что считать актерами и случаями использования, сколько случаев использования должно в среднем быть в спецификации системы и т.д. В каждом из этих вариантов используются одни и те же диаграммы случаев использования UML, однако их семантика и прагматика отличаются.

Подобные вариации можно назвать заготовками для различ-

ных визуальных языков проектов: они по-разному конкретизируют стандартный UML, отражая различный практический опыт своих создателей. Именно этот опыт, а также особенности проекта и создают ту "прослойку" между стандартными средствами визуального моделирования и тем, как они в итоге используются. При этом опыт аналитиков эволюционирует во времени.

Таким образом, практическое применение подобных индивидуальных наработок их авторами происходит через визуальные языки проектов. Такие наработки не просто передать для использования другим разработчикам (т.е. отделить от их авторов), поскольку они сильно "привязаны" к психологическим особенностям своих создателей, их мировосприятию, образованию и т.д. Однако будучи результатом творческого акта, они способны инициировать творческие находки (оказывать косвенное влияние, допуская не прямое, а опосредованное переиспользование). Анализ и проектирование невозможны без таких находок — например, книга Фаулера, посвященная практическому применению UML [11], буквально пронизана азартом и настроением творческого поиска.

Когда индивидуальная находка оказывается чем-то большим, чем просто эффективным инструментом ее автора, происходит рождение нового метода, нового элемента стандартного языка визуального моделирования и т.д. То есть опыт автора не только переходит в визуальные языки различных проектов, но становится частью стандартных средств визуального моделирования. Это произошло, например, с базовой концепцией случаев использования [17].

Визуальный язык проекта, используемый как средство коммуникации, обычно определяется только созданными с его помощью текстами[11]. Бывает также, что он описывается в неформальном виде, как в приводимом выше примере. Однако целесообразно создавать отдельное краткое описание такого языка в виде документа и в дальнейшем следовать этому документу, поддерживая его актуальность.Другими словами, такое описа-

[11]Подобным образом, единственной спецификацией многих древних языков является набор дошедших до нас текстов, написанных на этих языках [3].

ние нужно сделать элементом конфигурационного управления [22]. Это упорядочивает процесс моделирования и способствует согласованной работе в команде.

Приведем пример визуального языка проекта, созданного на основе UML и используемого как средства коммуникации в проекте по разработке промышленной информационной системы. UML применялся следующим образом: использовались диаграммы случаев использования, деятельностей и классов.

С помощью диаграмм случаев использования строились:

- контекстная модель системы;
- модель функций системы.

Прагматика диаграмм случаев использования по-разному уточнялась — отличался процесс их построения и применения.

С помощью activity-диаграмм была построена модель бизнес-процессов новой системы. Эти диаграммы использовались стандартным образом.

С помощью модели классов были построены:

- Концептуальная модель предметной области — связное и доступное описание основных понятий предметной области, имеющих принципиальное значение для построения информационной системы и в силу этого требующих детального согласования с заказчиком/пользователями системы. Эта модель представляла собой небольшой набор диаграмм, снабженных подробным текстовым описанием.
- Логическая модель предметной области — полная модель предметной области, необходимая для создания базы данных, а также требующая согласования со специалистами предметной области. Не содержала типов атрибутов, справочников, не были раскрыты отношениями вида “многие ко многим”, многие атрибуты были объединены в группы. Эта модель, в отличие от предыдущей, не предназначалась для широкого обсуждения.
- Физическая модель базы данных системы — полная спецификация схемы базы данных с учетом специфики целевой СУБД.

Как и в случае с диаграммами случаев использования, семантика и прагматика диаграмм классов менялась и варьировалась. Например, класс обозначал сущность предметной области и таблицу базы данных, использовалось три различных варианта прагматики диаграмм классов.

5 Визуальное моделирование с применением автоматической обработки

Одним из эффективных способов автоматизированного применения визуального моделирования является создание технологического решения [2]. Разработка и эволюция таких решений происходит, как правило, по следующей схеме. Сначала технологическое решение создается для решения какой-либо проблемы конкретного процесса[12]. После того как данный процесс разработки ПО завершается, технологическое решение переиспользуется, модифицируется и развивается в рамках других процессов этой же компании. Это происходило, например, с технологическими решениями компании ЛАНИТ-ТЕРКОМ — RTST [6] и REAL-IT [9], которые далее в этой работе приводятся в качестве примеров, а также с технологическими решениями многих других компаний. Визуальный язык проекта, являясь в этом случае составной частью технологического решения, также изменяется и развивается от одного проекта к другому (а не создается каждый раз заново).

Формализация визуального языка проекта в описанной ситуации необходима, поскольку этот язык реализуется[13] программными средствами технологического решения, являясь, таким образом, частью спецификации этих средств. От того, насколько полна спецификация создаваемых программных продуктов, зависит эффективность их реализации, дешевизна разработки и сопровождения.

Ниже приводятся примеры визуальных языков проектов, созданных в контексте технологических решений.

[12] В данной работе мы отождествляем понятия процесса и проекта, хотя между ними существуют различия проект — это создание нового сервиса или продукта [12], а процесс — нет.

[13] Например, создаются генераторы кода, валидаторы визуальных моделей проекта и т.д.

5.1 Пример 1

Технологическое решение RTST [6] было создано для преодоления ряда проблем большого телекоммуникационного проекта, среди которых была задача по наладке взаимодействия инженеров по оборудованию и программистов. В рамках проекта требовалось реализовать большое количество алгоритмов по управлению аппаратурой. Спецификации алгоритмов на языках программирования были трудны для понимания инженерами по оборудованию, поэтому было принято решение использовать визуальный язык SDL. Кроме того, было решено использовать автоматизированные генераторы для создания исходных кодов системы по SDL-моделям, чтобы графические спецификации не оставались просто картинками. При этом язык SDL был модифицирован следующим образом:

- использовалось только подмножество SDL, предназначенное для спецификации поведения, средства структурной декомпозиции SDL не применялись, а все данные описывались на цеоевос языке программирования (усечение синтаксиса);
- код в SDL-символах создавался на целевом языке программирования (изменение синтаксиса);
- не использовались SDL-процедуры, так как была необходима экономия памяти при реализации SDL-процессов — SDL-процедуры требуют хранения нефиксированного по длине стека вызовов (усечение синтаксиса);
- в качестве средств синхронизации процессов использовались только сигналы — т.е. не использовались удаленные процедуры, export- и view-переменные (усечение синтаксиса);
- использовалась псеводпараллельная модель взаимодействия процессов SDL (изменение семантики);
- допускалось создание циклов в переходах без использования соединителей (расширение синтаксиса).

Был создан собственный генератор кода, который транслировал модельную спецификацию в целевую платформу с учетом ее

специфических ограничений — небольшого размера сегмента кода, необходимости оптимизировать межсегментные вызовы, генерировать загрузочные модули различных типов и т.д.

Отметим, что прагматика полученного языка не изменилась — его использование является традиционным для SDL.

5.2 Пример 2

Технологическое решение REAL-IT [9] было создано для разработки информационных систем с использованием автоматизированной генерации базы данных и пользовательского интерфейса по логической модели данных. Для создания этой модели использовались диаграммы классов и кооперации UML, выполненные в CASE-пакете REAL [10]. Модель классов UML была модифицирована следующим образом:

- добавлена новая конструкция “Представление” (“View”), соответствующая одноименной конструкции в SQL (расширение синтаксиса);
- только абстрактные классы могли иметь наследников (усечение синтаксиса);
- не использовались связи “многие-ко-многим” (усечение синтаксиса);
- классы модели данных содержались в отдельных пакетах (усечение синтаксиса).

Диаграммы кооперации использовались для спецификации

- ограничений на связи между классами (изменение семантики).
- предопределенных объектов в базе данных (уточнение семантики).

В обоих случаях использовались специальные стереотипы для объектов и связей между ними.

6 Автоматизированная валидация синтаксиса визуальных спецификаций

При применении к визуальным спецификациям средств автоматической обработки встает проблема контроля созданных спецификаций на соответствие правилам визуального языка проекта. Ниже рассказывается о методе спецификации синтаксиса визуального языка проекта с помощью языка OCL с последующим автоматическим контролем (валидацией) визуальных спецификаций проекта на соответствие этому синтаксису[14].

6.1 Спецификация визуального языка проекта с помощью OCL

В 1994 г. в рамках подхода Syntropy [14] предложен текстовый язык для уточнения визуальных спецификаций. Однако этот язык, основанный на математических абстракциях, сложен для использования. Корпорация IBM создала на основе подхода Syntropy язык OCL[15] [21], который был включен в стандарт UML.

Язык OCL предназначен для спецификации дополнительной информации в моделях, которую неудобно (невозможно) выражать графическим способом. Предлагается использовать его также для спецификации операций усечения синтаксиса языка[16]. Такие операции, выраженные с помощью OCL, будем называть OCL-ограничениями. Ниже приводится фрагмент OCL-ограничений для визуального языка проекта, созданного в рамках технологического решения REAL-IT [9].

Данное ограничение заключается в запрете использования связей вида “многие ко многим”. Оно также связано с особенностями генерации баз данных в REAL-IT. На рисунке 1 представлен соответствующий фрагмент метамодели UML.

[14] Более подробно этот метод изложен в работе [5].

[15] Object Constraint Language.

[16] С помощью средств расширения UML — стереотипов (stereotypes) и прекрепленных значений (tagged values) — можно добавлять новые конструкции и новые атрибуты стандартным констукциям. На OCL можно “сужать” использование как стандартных конструкций, так и вновь созданных.

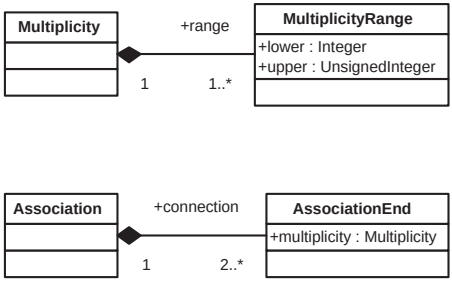

Рис. 1: Фрагмент метамодели

Спецификация этого OCL-ограничения выглядит следующим образом:

```
context Association
inv: self.allConnections -> select(
  multiplicityRange -> select( upper > 1 ) -> notEmpty
                            ).size < 2
```

6.2 Метод и его программная поддержка

Организация поддержки визуального языка проекта происходит следующим образом:

- описание визуального языка проекта при помощи OCL-ограничений;
- генерация валидатора по OCL-ограничениям;
- интеграция валидатора в процесс производства ПО.

Последний этап является существенным — только при регулярном контроле корректности визуальных моделей (например, при использовании систем “ночной сборки” и регрессионного тестирования) будет извлекаться максимальная выгода, так как лишь в этом случае ошибки будут обнаруживаться и исправляться до того, как они нанесут существенный ущерб всему процессу. Архитектура валидатора и генератора валидаторов подробно описывается в работе [5].

6.3 Результаты апробации

По итогам апробации валидатора были сделаны следующие выводы относительно случаев эффективного его использования.

1. При контроле ограничений, которые либо больше нигде не проверяются автоматически (например ограничения по стандартному оформлению диаграмм), либо проверяются лишь на поздних этапах (например как ограничения на именование идентификаторов, нарушение которых будет выявлено только при компиляции сгенерированного кода).

2. В случае, когда автоматической обработке визуальной модели предшествует длительный процесс собственно моделирования. В этой ситуации имеет смысл как можно раньше наладить базис ("base-line") [16] и поддерживать ее в том числе с помощью регулярно запускаемого в пакетном режиме валидатора (в стиле регулярной сборки [8]). Иначе, как показывает опыт, установка процесса генерации, компиляции сгенерированного кода и его отладки становится очень трудоемкой.

3. В случае, когда при разработке в модель вносятся крупные изменения, например, вследствие реализации дополнительной функциональности. В этой ситуации изменения могут вноситься не только автором соответствующего фрагмента модели, поэтому оказывается важным как можно раньше контролировать случаи нарушения целостности визуальной модели.

Заключение

На данный момент не существует универсального процесса разработки ПО. Это связано со значительными индивидуальными особенностями программных проектов, а также из-за творческого характера процесса разработки ПО. Создано большое количество различных моделей процессов, методов и методик, а также программных средств и линеек программных продуктов. Все эти разнообразные инструменты подразумевают, тем не менее, существенную работу по внедрению в конкретный процесс. Визуальное моделированиие не является исключением. В связи с

этим возникает протребность в методах поддержки визуальных языков проектов.

Синтаксис является отношением конструкций языка между собой, поэтому его соблюдение легко проверяется автоматически. Метод проверки синтаксиса визуального языка проекта изложен в этой работе. Семантика и прагматика определяют отношения текста с внешними сущностями (объектами и интерпретатором), поэтому затруднительно автоматически проверять их корректность только на основе текста.

Отсутствие ясной, однозначно определенной семантики стандартных визуальных языков затрудняет их унифицированное использование. Вариации семантики конструкций стандартных визуальных языков, по всей видимости, возможно специфцировать только в виде документов или в рамках специальных программных средств технологического решения.

В наибольшей степени подвержена вариативности прагматика стандартных визуальных языков. Именно там, в основном, выражаются индивидуальные особенности создаваемой программной системы и процесса ее разработки. Преобладание вариативности прагматики над вариативностью других частей языка (синтаксиса и семантики) в некоторых случаях может поставить под сомнение применимость семиотического подхода при исследовании особенностей использования визуального моделирования в конкретных проектах. Однако важность такого подхода заключается в том, что он ведет к формализации и фиксации создаваемых диалектов. Иначе, как показывает практика, люди незаметно перестают понимать создаваемые друг другом диаграммы, а сами диаграммы становятся непонятны даже их создателям при возврате к ним после достаточно длительного перерыва. Возникает также большое количество ошибок в специальных программных средствах технологических решений, которые трудно исправимы из-за неточных спецификаций.

Необходимо отметить, что визуальные языки проекта обычно не создаются целиком в рамках одного проекта, а имеют более сложную историю. В большинстве случаев они основываются на предыдущем опыте компании и аналитиков данного проекта. Этот опыт, как правило, трудно отчуждаем от его носителей — например, почти все известные нам технологические решения, созданные в разных компаниях, использовались только там, а

попытки переноса их в другие компании терпели неудачи.

В заключение еще раз подчеркнем, что создание визуальных языков проектов не является изобретением абсолютно новых средств моделирования, а происходит на основе существующих стандартных визуальных языков.

Список литературы

[1] Ван Вейнгаарден А., Майу Б., Пек Дж. и др. Пересмотренное сообщение об Алголе-68. — М.: Мир, 1979. — 533 с.

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

[3] Лотман Ю.М. Текст в тексте // Статьи по семиотике культуры и искусства. — М.: Академический проект, 2002. — С. 58-58.

[4] Моррис Ч.У. Основания теории знаков // Семиотика — М.: Радуга, 1983. — С. 37-90.

[5] Ольхович Л., Кознов Д.В. Метод автоматической валидации UML-спецификации на основе языка OCL // Программирование. — № 6. — 2003. — С. 44-50.

[6] Парфенов В.В., Терехов А.Н. RTST — технология программирования встроенных систем реального времени // Системная информатика. — Вып. 5. — 1997. — С. 228-256.

[7] Поттосин И.В. Программная инженерия: содержание, мнения и тенденции // Программирование. — № 4. — 1997. — С. 26-37.

[8] Соммервилл И. Инженерия программного обеспечения. — М.: Вильямс, 2002. — 624 с.

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

[10] Терехов А.Н., Романовский К.Ю., Кознов Д.В. и др. Real: Методология и CASE-средство для разработки систем реального времени и информационных систем // Программирование. — № 5. — 1999. — С. 44-52.

[11] Фаулер М., Скотт К. UML. Основы. Краткое руководство по унифицированному языку моделирования. — М.: Символ-Плюс, 2002. — 192 с.

[12] A Guide to the Project Management Body of Knowledge (PMBOK Guide). — Pennsylvania: Project Management Institute, 2000. — 216 p.

[13] Boulychev D., Koznov D.V., Terekhov A.A. On project-specific languages and their application in reengineering // Proc. of Conference on Software Maintenance and Reengineering. — 2002. — P. 177-185.

[14] Cook S., Daniels J. Designing Object Systems: Object-Oriented Modelling with Syntropy. — Prentice Hall: 1994. — 388 p.

[15] http://www.microsoft.com/office/visio/

[16] Humphrey W.Managing the Software Process. — Boston: Addison-Wesley, 1989. — 512 с.

[17] Jacobson I., Christerson M., Jonsson P. Object-Oriented Software Engineering: A Use Case Driven Approach (1st ed.). — Boston: Addison-Wesley, 1992. — 528 p.

[18] Goldstine H., Von Neumann J. Planning and Coding Problems for an Electronic Computing Instrument // Papers of Jonhn von Neumann on Computing and Computer Theory. Part II. Vol. 1. — MIT Press: 1947. — P. 151-223.

[19] Marca D.A., McGowan C.L. SADT Structured Analysis and Design Technique. — McGraw-Hill: 1988. — 392 p.

[20] Mehandjiska-Stavreva D., Page D., Choi M. Meta-modelling and Methodology Support in Object-Oriented CASE tools // Proceedings of the OOIS'96. — 1996. — P. 370-383.

[21] OMG Unified Modeling Language Specification. Version 1.4. — 2002. — http://www.uml.org.

[22] Paulk M.C., Garcia S.M., Chrissis M. e.a. Key Practice of Capability Maturity Model, Version 1.1: Technical Report CMU/SEI-93-TR-025 ESC-TR-93-178. — 1993. — 479 p.

[23] Raeder G. A Survey of Current Graphical Programming Techniques // IEEE Computer. — Vol. 18. № 8. — 1985. — P. 11-25.

[24] Ross D. Structured Analysis (SA): A Language for Communicating Ideas // IEEE Transactions on Software Engineering. — Vol. 3. № 1. — 1977. — P. 16-34.

[25] Specification and Description Language — ITU-T-Z.100, 2002. — 203 p.

[26] Yourdon E. Modern Structured Analysis. — Englewood Cliffs, NJ: Yourdon Press, 1989. — 672 p.