ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г.

Журнал издается при поддержке Отделения математических наук РАН, Отделения нанотехнологий и информационных технологий РАН, МГУ имени М.В. Ломоносова, МГТУ имени Н.Э. Баумана, ОАО "Концерн “Сириус"

СОДЕРЖАНИЕ

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в любом почтовом отделении (индексы: по каталогу агентства "Роспечать" — **22765**, по Объединенному каталогу "Пресса России" — **39795**) или непосредственно в редакции.
Тел.: (499) 269-53-97. Факс: (499) 269-55-10.
Http://novtex.ru/pi.html E-mail: prin@novtex.ru
Журнал включен в систему Российского индекса научного цитирования.
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

№ 5
May
2015

Published since September 2010

CONTENTS

Information about the jornal is available online at:
http://novtex.ru/pi.html, e-mail:prin@novtex.ru

УДК 519.683

И. Якобсон, проф., президент SEMAT, e-mail: ivar@ivarjacobson.co,
Э. Сейдевитц, технический директор, Ivar Jacobson International, Швейцария
(редактор перевода — д-р техн. наук, проф. **Б. А. Позин**, e-mail: bpozin@ec-leasing.ru)

Новая программная инженерия

Рассмотрены недостатки существующего состояния программной инженерии, представлены основные идеи и направления воссоздания программной инженерии как инженерной дисциплины. Сформулировано ядро SEMAT — минимальный набор сущностей методологии, или метода (так называемых альф), и отношений между ними, определена совокупность характеристик состояния работ по созданию целевого программного продукта с заранее заданными требованиями к нему. Обеспечена возможность систематической оценки "степени прогресса" проекта и "состояния здоровья" деятельности по его реализации. Показаны пути поэтапного развития программной инженерии как прикладной науки.

Ключевые слова: *SEMAT, ядро SEMAT, альфа, контрольные состояния альф, метод, практика, программная система, области интересов, контрольные карты*

Введение

Что произошло с программной инженерией? Что стало с обещанием, которое давали инициаторы SEMAT [1] относительно разработки строгих, упорядоченных, профессионально описанных практик (знаний и опыта, основанных на результатах практической деятельности[1]), направленных на разработку программного обеспечения, подобно тому, как они освещаются в других инжиниринговых дисциплинах? Действительно, то, что было принято понимать под "программной инженерией", представляет набор практик, которые в основном адаптированы из других инженерных дисциплин. К их числу относятся: управление проектами, проектирование и планирование, управление процессами разработки и т. д. Основным признаком, объединяющим эти практики, является рассмотрение программного обеспечения как продукта производства, со всеми реальными этапами инжиниринга в общем понимании этого вида деятельности, а именно с анализом требований, проектированием, моделированием и т. д.[1]

Следует отметить, что выполнять работы таким путем в других инженерных дисциплинах целесообразно, потому что первичная работа основывается на годами накопленных фундаментальных знаниях, на единой методологической основе, что позволяет доверять их положениям. Программная инженерия относительно новая область знаний, она не имеет такой богатой практиками базы. При этом следует заметить, что затраты на реализацию проектов по созданию фундаментальных основ программной инженерии в настоящее время, как правило, не окупаются. Более того, в традициях системной инженерии присутствует склонность понижать роль программистов-кодировщиков. Такая тенденция реализуется, если не напрямую, то через механизмы управления практиками. Однако программисты-кодировщики — это те специалисты, которые на практике должны выполнить работу по созданию программного кода, которую они *выполняют* вне зависимости от того, соответствует ли результат проекту или нет. Не удивительно, что сложившееся положение дел привело к большой неудовлетворенности со стороны программистского сообщества.

На настоящее время развитие и систематизация эвристических знаний и практических навыков, направленных на создание программного обеспечения, представляет собой содержание инженерного подхода. В контексте данной статьи будем именовать такой подход *ремесленным*.

Если при разработке программного обеспечения сосредоточить внимание на таком (ремесленном) подходе, то с позиции развития вполне правомерен вопрос: "Целесообразен ли вообще *инжиниринг* программного обеспечения?" Действительно ли такая точка зрения разумна?

Так как в результате выполнения проекта должен быть реализован программный код, представляется разумным с самого начала сосредоточить внимание на отработке качественного кода. Кодирование является ремесленной дисциплиной, как следствие, оно может основываться на опыте "мастеров" программирования, которые вносят определяющий вклад в методологию создания качественного кода. Например, многие технические методы гибкой разработки кода

[1] Уточнение редактора-переводчика.

(*agile development*) к настоящему времени позволили создать высококачественные системы программного обеспечения значительного размера с использованием ремесленного подхода к его реализации. И это — несмотря на отрицание определяющей роли любых действий, которые соответствуют современному взгляду на инжиниринг программного обеспечения.

Следует, однако, отметить, что методология разработки программного обеспечения, основанная на ремесленном подходе, может привести к решению только ограниченного класса задач. С древних времен и во времена Средневековья квалифицированные ремесленники и мастера создали много чудесных строительных сооружений — от пирамид до готических соборов. К сожалению, строительство этих сооружений было невероятно дорогим и трудоемким, тем не менее иногда они разрушались по не совсем понятным причинам.

Создание современных строительных сооружений, таких как небоскребы, стало возможным только с разработкой "правильного" *инженерного* подхода (методов инжиниринга). У современного строительства есть твердая основа в материаловедении и теории строительства, а инженеры-строители используют эту теоретическую базу в качестве основы для точного, упорядоченного подхода к проектированию зданий и сооружений, которые они должны построить.

Конечно, подобные здания и сооружения и в настоящее время иногда подвергаются разрушениям. Однако, когда такое происходит, снова проводится полный анализ ситуации с тем, чтобы определить, была ли неудача вызвана неправомерными действиями строителей, или недостатки кроются в основной теории, которая использовалась в оригинальном проекте. Как следствие, в последующем новое осмысление причин может быть включено в основополагающую практику и будущую теоретическую базу.

Строительство является примером того, как реальная техническая дисциплина объединяет профессиональное мастерство с прикладной теоретической базой. Анализ и осмысление тех или иных ситуаций, которые фиксируют и принимают как методологическую базу, используют для обучения специалистов, начинающих осваивать ту или иную техническую дисциплину. Специалистам предоставляется основа, включающая базовые положения, для того чтобы системно анализировать и решать инжиниринговые задачи, даже когда эти задачи лежат за пределами опыта инженеров-практиков.

С этих позиций программная инженерия *пока еще не является в полной мере инженерной дисциплиной*. Как следствие, необходима "новая" программная инженерия, основанная на опыте ремесленников программного обеспечения, объединяющая их понимание в единую методологическую основу, которая затем может быть использована для обучения и поддержки нового поколения практиков. Поскольку ремесло основывается на практике, а весь смысл инженерной теории состоит в поддержке практиков, такой подход может восполнить то, что было упущено в предыдущих положениях программной инженерии.

Возникает вопрос: "Как сообществу программной инженерии решить задачу "воссоздания" программной инженерии"? Инициатива SEMAT (Software Engineering Method and Theory — теория и методы программной инженерии) — это попытка на международном уровне взаимодействия найти ответ на этот вопрос (см. http://www.semat.org). Как следует из названия, SEMAT концентрирует свои усилия на поддержке и обобщении опыта, основанного на ремесленном подходе (практические *методы*) и на создании базового знания (*теория*). Эта работа пока находится на начальной стадии. Однако суть новой программной инженерии становится все более и более понятной. Далее в статье анализируются ответы на вопросы, что представляет собой программная инженерия и каковы ее последствия для будущего этой дисциплины.

Инженерия — ремесло, поддерживаемое теорией

Метод (*методология*) представляет собой описание технологии работ по проведению вида деятельности, направленной на достижение определенного результата. Такой деятельностью является разработка программного обеспечения как продукта. В целом все методы являются производными от опыта человека-практика, который сначала перерабатывается в эмпирические правила и знания, а затем аккумулируется в методологии, направленной на реализацию предметной деятельности. В конечном счете при наличии консенсуса эти методы превращаются в стандарты.

В ремесле методы в значительной степени развиваются за счет опыта мастеров, которые в обязательном порядке обладают многолетним опытом работы по предмету деятельности. В старые времена мастера хранили свои методы втайне и передавали их только доверенным ученикам. В современном мире различные подходы, основанные на опыте работы таких мастеров, как правило, широко распространяются, в том числе путем публикации результатов.

Поскольку ремесло перерастает в инженерную дисциплину, важно выявить в методах различных мастеров ее осмысление и понимание, основанное на опыте, полученном в процессе осуществляемой ими той или иной предметной деятельности. Такое общее понимание включается затем в теоретические положения, которые можно использовать как основу различных методов для их применения в профессиональной деятельности.

В этом смысле теория — это не "плохое" слово, как иногда она трактуется в нашей культуре, например: "О, это просто теория". Как отмечалось ранее,

наличие теоретической основы является, фактически, ключом, который позволяет системным образом выполнить инженерный анализ, необходимый по предмету деятельности. Такая теоретическая основа создана в различных областях инженерной деятельности: материаловедение — основа для строительной техники, электромагнитная теория — основа для электротехники, аэродинамика — основа для авиационной техники и т. д.

Конечно, взаимосвязь между историческим развитием инженерной дисциплины и связанной с ней теорией, как правило, сложнее, чем предложенное выше объяснение. Инженерный опыт преобразуется в теорию, которая затем способствует более качественному инжинирингу, и так далее — по спирали. Тем не менее важный момент, который необходимо осознать, — это то, что традиционной программной инженерии недоставало базовой теории как таковой.

Можно было бы предположить, что информатика обеспечивает теоретическую базу для программной инженерии. И вполне вероятно, что это было исходным ожиданием для специалистов в области разработки программ, когда программная инженерия впервые была задумана. Однако в действительности информатика осталась в основном академической дисциплиной. Ее положения сосредоточены на науке о вычислениях в целом, однако главным образом на науке, которая далека от создания методов программной инженерии в индустрии. В то время как "формальные методы" информатики обеспечивают перспективы некоторого полезного теоретического анализа программного обеспечения, практики в основном избегают таких методов (кроме нескольких специализированных областей, таких как численные методы).

В результате часто возникали соревновательные циклы различных методологий программной инженерии без наличия основополагающей теории для их объединения. В итоге многие из этих методов даже не отражали реальных потребностей квалифицированных практиков, работающих в сфере создания промышленного программного обеспечения.

Итак, как в сложившейся ситуации поступить?

Создание законченной, новой теории программной инженерии займет некоторое время. Вместо того чтобы начинать с академического подхода, можно начать, как уже упоминалось, с выявления общности между методами, которые оказались успешными в ремесленном подходе к разработке программного продукта. Выявление общности требует создания общего подхода к описанию, осмыслению, и объединению различных методов разработки программного продукта вместо создания основы для конкуренции этих методов друг с другом. Чтобы понять, как конечная цель может быть достигнута, рассмотрим методы и реализующие их команды практиков, которые используют такие методы в реальной работе.

Гибкость методов, а не только программного обеспечения

Действия, направленные на достижение гибкости в разработке программного обеспечения, не только не противоречат, а являются дополнением к уже существующим знаниям и опыту, которые получены при ремесленном подходе. Как следует из названия, гибкая (*agile*) разработка программного продукта — это технология, которая обеспечивает высокий уровень приспособляемости и адаптируемости к среде разработки в условиях объективно изменяющихся требований к конечному продукту. Это делается путем пошагового анализа состояния программного продукта, получением быстрой обратной связи для корректировок, если в них возникает необходимость.

Команды, реализующие agile-разработку программного продукта, берут на себя ответственность за используемую ими методологию. Такая команда применяет методы, которые она считает нужными в своем проекте, адаптируя процесс разработки программного продукта к изменяющимся требованиям к нему на протяжении всего времени выполнения проекта. В действительности, agile-команда должна развивать и совершенствовать свои методы так же гибко, как она осуществляет разработку программного продукта.

Здесь необходимо отметить, что отсутствие гибкости в методах является главным недостатком традиционной программной инженерии.

Программное обеспечение по природе своей "податливо" и легко (физически) изменяемо. Сложная программная система, однако, может показать своего рода "интеллектуальную жесткость". В такую систему трудно внести изменения без последующих потерь. Каждое изменение в системе порождает иногда даже большее число ошибок, чем те, которые оно устраняет. На этом фоне ответом традиционному подходу к разработке программного обеспечения могло бы стать принятие методов контроля процессов управления проектами, аналогичных тем, которые используют при решении подобных задач в сложных аппаратных системах.

Однако с точки зрения agile-разработки, для программного обеспечения применение методов, характерных для проектирования аппаратуры, может быть ошибкой. Причина в том, что методики agile используют изменяемую природу программного обеспечения, а также возможности быстрого анализа состояния разработки для получения обратной связи. Это обеспечивается в ходе перманентного мониторинга состояния разрабатываемого продукта путем его комплексного тестирования. Целью при этом является решение вопросов, обусловленных сложностью программного продукта, а не контроль процесса разработки. В результате гибкая разработка ориентирует специалиста-практика на создание качественного программного продукта, а не на то,

чтобы он поддерживал наперед заданный процесс разработки.

Таким образом, ответ на вопрос, как гибкость вводится в методы программной инженерии, в самом общем виде может быть сформулирован следующим образом: "Через анализ и осознанную оценку специалистами-практиками основных действий, которые они фактически предпринимают, а именно — через их практики (приемы работы)".

Методы возникают из практик

На первый взгляд, метод может казаться монолитным. Однако любой метод может рассматриваться, как состоящий из ряда практик. Практика представляет собой осознанно повторяемый подход к выполнению действий с определенной целью. Другими словами, практики — это то, что фактически осуществляют специалисты в своей деятельности.

В качестве примера можно привести гибкий метод экстремального программирования, который описан как 12 практик, включающих в себя парное программирование, разработку через тестирование и непрерывную интеграцию.

Agile-платформа Scrum включает такие практики, как поддержание списка требований, ежедневные совещания и спринты. С этих позиций Scrum в действительности является не законченным методом, а составной практикой, которая построена как композиция ряда других практик, взаимодействующих друг с другом. При этом Scrum может использоваться в качестве базовой платформы процесса разработки программного обеспечения в сочетании с практикой, например, экстремального программирования. Как результат, формируется новый метод, который может использоваться agile-командой разработчиков.

Представленный пример иллюстрирует, каким образом методы составляются из практик. Команды могут собрать воедино практики, которые наилучшим образом соответствуют задаче конкретной разработки, а также адекватны навыкам членов команды разработчиков. Кроме того, при необходимости команда может развивать свой метод не только за счет малых изменений, но и за счет более радикальных и больших шагов. Таким шагом может быть, например, замена старой практики на более хорошую новую без изменения каких-либо других составляющих практик.

Важно, что в процессе разработки акцент должен делаться на команды и отдельных практиков в командах, а не на "инженеров по методам", которые целенаправленно создают методы для реализации их другими людьми. Создание своей собственной методологии (технологии) работы является первоочередной и новой обязанностью для большинства команд разработчиков. Следует также иметь в виду то обстоятельство, что необходимо поддерживать способность команд делать это в разных проектах. Такие подходы особенно полезны для групп, заинтересованных в создании новых и расширении уже сложившихся практик, которые могут использоваться вне определенного проекта. Такие практики в случае необходимости могут быть использованы другими проектными командами.

Перечисленные выше соображения можно рассматривать как сочетание интересов на разных уровнях формирования и применения практик. Практики могут быть созданы и систематизированы в пределах отдельной организации-разработчика или в рамках промышленных групп, состоящих из нескольких организаций (что, например, эффективно в случае с Extreme Programming и Scrum). Специалисты-практики в проектных командах могут перенимать, адаптировать и применять такие рекомендуемые им практики.

Вполне естественен вопрос: "Какие у проектных команд есть гарантии того, что разрозненным образом созданные практики могут на самом деле быть без особых усилий объединены для получения ими эффективных методов?" Такой гарантией, по мнению авторов, является то обстоятельство, что в среде специалистов созрело понимание необходимости создания новых основ программной инженерии, независимых от практик и методов, однако способных создать для них общую методологическую платформу.

Ядро — основа для практик и методов

Первым существенным результатом инициативы SEMAT, которая направлена на формирование положений современной программной инженерии [2], является так называемое ядро для разработки программного обеспечения. Это ядро можно рассматривать как минимальный набор элементов, которые являются универсальными для всех специалистов в сфере разработки программного обеспечения. Ядро включает минимальный набор сущностей, которые универсальны для всех способов организации деятельности по разработке программного продукта. Ядро состоит из трех частей и включает:

- средства для измерения так называемых степени прогресса и состояния здоровья деятельности коллектива (направленной на создание продукта);
- категоризацию действий, которые необходимо выполнить для того чтобы увеличить прогресс и состояние здоровья целенаправленной деятельности в целом;
- набор компетенций, необходимых для проведения таких действий.

Под *степенью прогресса* и *состоянием здоровья* в SEMAT имеется в виду (понимается) совокупность характеристик состояния работ по созданию целе-

вого программного продукта с заранее заданными требованиями к нему.

Особое значение в контексте ядра имеют его базовые сущности. Они формируют четкое представление о том, как деятельность коллектива по разработке программного продукта прогрессирует. Ядро SEMAT определяет семь направлений для измерения достигнутого прогресса, известных как альфы. Термин *альфа* изначально являлся акронимом (ALPHA — Abstract-Level Progress Health Attribute) атрибута прогресса и здоровья на абстрактном уровне. Однако теперь он, как отмечено в ядре, используется для определения степени прогресса и состояния здоровья в разработке продукта. Было рассмотрено много других предлагаемых терминов, многие из которых имели смысл, который вступал в противоречие с принципиально новой концепцией, введенной для ядра. Как итог, был принят новый термин альфа, которого ранее не существовало. К числу семи альф ядра относятся: возможность; заинтересованные стороны; требования; программная система; работа; команда; технология работы. Эти альфы связаны друг с другом как показано на рис. 1 (см. вторую сторону обложки).

Каждая альфа имеет определенный набор состояний, в которых кодифицированы точки, позволяющие измерить состояние прогресса разработки, представленного той или иной альфой. У каждого из состояний есть контрольный список, призванный помочь специалистам контролировать текущее состояние деятельности по определенной альфе и понимать состояние, которого они должны добиться на следующем шаге. Идея состоит в том, чтобы обеспечить интуитивно понятный инструментарий для практиков, позволяющий судить о степени прогресса и о состоянии здоровья деятельности разработчиков общим, независимым от конкретно применяемых методов способом.

Одним из способов визуализации семимерного пространства альф является использование диаграммы типа "паутина" [3], которая показана на рис. 2 (см. вторую сторону обложки). На этой диаграмме закрашенная область показывает, какого прогресса удалось добиться в деятельности по разработке программного продукта. Незакрашенная область отражает то, что должно быть сделано для завершения деятельности по разработке. Беглый взгляд на подобную схему обеспечивает хорошее представление о том, где (в каком состоянии) проект находится в текущий момент времени.

Альфы позволяют сделать отображение процесса разработки более удобным для восприятия путем помещения каждого из состояний альф на карте вместе с контрольным списком этого состояния в сокращенном виде (рис. 3, см. третью сторону обложки). Колода таких карт может легко поместится в карман человека. Несмотря на то что более подробные инструкции доступны, эти карты содержат ключевые напоминания, которые могут использоваться группами разработчиков в их ежедневной работе, так же как справочник инженера по другим дисциплинам.

Более полное обсуждение ядра и его применения можно найти в работах [1, 4]. Само ядро формально определено как часть спецификации Essence, которая была стандартизирована Object Management Group [5]. В дополнение к полному ядру стандарт Essence определяет язык, который может использоваться как для описания ядра, так и для описания практик и методов в терминах ядра. Важно отметить, этот язык предназначен для того, чтобы его могли применять практики, а не только инженеры по методам. Для использования по назначению он может быть изучен в течение двух часов (карты состояний альф — простой пример этого).

С учетом изложенного выше можно сделать вывод о том, что возможность использовать ядро для описания практик позволяет создать необходимую методологическую платформу для формирования на ее основе правильных (перспективных) методов программной инженерии.

Практики, основанные на ядре, создают условия для описания гибких методов.

Практика может быть выражена в терминах ядра через:

— выявление областей, в которых она улучшает деятельность;

— описание действий, которые нужно провести для необходимого продвижения и разработки соответствующих рабочих продуктов;

— описание специфических компетенций, необходимых для осуществления определенных действий.

Практика может также расширить ядро с помощью дополнительных состояний, контрольных списков или даже новых альф. Важным моментом является тот факт, что ядро обеспечивает общий каркас для описания всех практик и позволяет объединять их в методы. Приведение набора практик в общую систему позволяет более четко идентифицировать разрывы и перекрытия в их применении. Разрывы могут быть заполнены с помощью дополнительных практик, а перекрытия устраняются путем соответствующего объединения дублирующих друг друга практик.

В качестве примера рассмотрим две практики: одна — использование списка требований (*backlog*) для управления работой, которая будет выполнена командой (совершенствование альфы "Работа"); другая — определение требований на основе пользовательских историй (совершенствование альфы "Требования"). Практика составления списка требований не предписывает, какие элементы должны быть в его составе. Практика пользовательских историй не предписывает, как команда должна управлять реализацией этих историй. Таким образом, эти две

практики дополняют друг друга и могут быть использованы вместе. Однако при таком "механическом" объединении они дублируют друг друга. Две практики могут быть соединены плавным и интуитивно понятным способом в общий метод путем включения в список элементов пользовательской истории как элементов в управляемый или другой список требований.

В частности, обратите внимание, как общая платформа ядра обеспечивает возможности прогнозирования. Инженер-строитель может использовать материаловедение и строительную механику, чтобы на ранней стадии понять, будет ли предложенное здание с высокой вероятностью долго стоять или скоро вполне вероятно может разрушиться. Аналогично, используя ядро, разработчик программного обеспечения может понять, является ли предлагаемый метод хорошо сконструированным или в нем есть разрывы или дублирования практик, а также как разрешить возникшие вопросы.

Кроме того, через разделение интересов, которое обсуждалось ранее, организация или сообщество могут создать библиотеку практик и даже основных методов, на которые новая проектная команда может опираться при формировании ее начальной технологии работы. Каждая команда тогда может продолжать гибко адаптировать и развивать свои собственные методы в рамках общего каркаса Essence [6].

В итоге, целью индустрии будет являться стандартизация особенно полезных и успешных практик. В то же время такой целью является повышение, а не ограничение гибкости команд в применении и адаптации этих практик, а также создание новых практик, по мере необходимости. В конечном счете, это реальный путь к созданию отвечающей современным требованиям программной инженерии.

Заключение

В наши дни, возможно, несколько злоупотребляют термином "сдвиг парадигмы". Тем не менее основанный на ядре Essence подход к разработке программного обеспечения вполне обоснованно можно считать таким сдвигом. Он действительно представляет собой существенное изменение точки зрения программно-инженерного сообщества.

Когда Томас Кун представлял понятие парадигмы в своей нашумевшей книге [7], он подчеркнул сложность (Кун даже утверждал — невозможность) перевода языка и теории одной парадигмы в другую. Сообщество разработчиков программного обеспечения фактически ранее видело подобные сдвиги. При этом те, кто погружен в старую парадигму, испытывают затруднения даже при понимании того, что представляет собой парадигма новая. Переход к объектной ориентированности был таким же сдвигом, как и во многих отношениях сдвиг в настоящем к agile-методам.

В связи с этим обстоятельством Essence действительно может рассматриваться, как сдвиг парадигмы в двух направлениях. Во-первых, те, кто погружен в "старую школу" программной инженерии, должны начать думать, в частности, о "правильной" программной инженерии, вместо того чтобы просто применять практики, в основном адаптированные из других технических дисциплин. Во-вторых, те, кто связан с ремесленным подходом к разработке программного обеспечения и с agile-сообществами, должны увидеть развитие "правильной" инженерной дисциплины как необходимого этапа эволюции от их прикладной дисциплины (совсем недавно с трудом сформированной).

Что касается второго пункта, в предисловии к работе [1], Роберт Мартин, один из тех, кто согласился с инициативой SEMAT, описывает "классическое колебание маятника" от программной инженерии в направлении к ремесленному подходу к разработке программного обеспечения. Оценка Мартина корректна, однако важно отметить, что этот пресловутый "маятник" не должен просто отклоняться назад в направлении, откуда он пришел. Наоборот, во время колебания он должен, и сейчас это необходимо, качаться с разницей едва ли не в 90 градусов от того направления, откуда он пришел, с тем чтобы переместиться к новой дисциплине — "правильной" программной инженерии.

Едва ли существует, пожалуй, лучшее представление (интерпретация) сдвига парадигмы, чем изложенная выше. В итоге новая парадигма программной инженерии, при режиме ее эволюции из текущей парадигмы ремесленного программирования, должна двигаться в сторону поэтапного отказа от положений старой парадигмы программной инженерии. Безусловно, как и все предыдущие сдвиги парадигмы, этот путь займет значительное время и будет стоить больших усилий, прежде чем будет завершен. Только после этого в качествах новой парадигмы все увидят ее очевидные преимущества.

Тем не менее даже в его нынешнем виде использование Essence может предоставить команде некоторые ключевые преимущества. Положения Essence помогают командам разработчиков быть гибкими при использовании методов и оценивать прогресс разработки с точки зрения ее реальных итогов и результатов, представляющих интерес для заинтересованных сторон. Такие измерения степени прогресса разработки оцениваются по семи направлениям — альфам ядра, которые все вместе должны продвигать ее в некотором темпе вперед, чтобы снизить риски и обеспечить нужный результат.

Кроме того, Essence может позволить организациям упростить управление методами путем использования пула практик, которые могут быть приняты

и адаптированы проектными командами. Наличие Essence в качестве общей основы для этого также позволяет специалистам-практикам учиться друг у друга с большей готовностью.

Реальный сдвиг, однако, произойдет только тогда, когда команды действительно используют все преимущества настоящего Essence и когда SEMAT построит на базе Essence новую парадигму программной инженерии. Сообщество практиков в настоящее время вносит свой вклад в движение на этом направлении и становится частью этого воссоздания программной инженерии.

Список литературы

1. **Jacobson I., Ng P.-W., McMahon P. E., Spence I., Lidman S.** The Essence of Software Engineering: Applying the SEMAT Kernel. Addison Wesley, 2013. 224 p.

2. **Пак Дж. С., Якобсон И., Майбург Б., Джонсон П.** SEMAT вчера, сегодня и завтра: перспективы промышленного использования// Программная инженерия. 2014. № 11. С. 6—16.

3. **Graziotin D., Abrahamsson P.** A Web-based modeling tool for the SEMAT Essence theory of software engineering // Journal of Open Research Software. 2013. Vol. 1, N. 1. e4; URL: http://dx.doi.org/10.5334/jors.ad

4. **Jacobson I., Ng P.-W., McMahon P., Spence I., Lidman S.** The Essence of software engineering: the SEMAT kernel // ACM Queue. 2012. Vol. 10, N. 10. URL: http://queue.acm.org/detail.cfm?id=2389616

5. **Object** Management Group. Essence—Kernel and language for software engineering methods. 2014. URL: http://www.omg.org/spec/Essence

6. **Jacobson I., Spence I., Ng P.-W.** Agile and SEMAT—Perfect Partners // Communications of the ACM 2013. Vol. 56, N. 11. P. 53—59. URL: http://cacm.acm.org/magazines/2013/11/169027-agile-and-semat/abstract

7. **Kuhn T.** The Structure of Scientific Revolutions. Chicago: University of Chicago Press, 1962. 264 p.

I. Jacobson, Professor, Chairman, e-mail: ivar@ivarjacobson.com,
E. Seidewitz, Chief Technology Officer, Ivar Jacobson International, Switzerland
(Editor-translator **B. A. Pozin**, Professor, MIEM of National Research University Higher School of Economics, Chairman, SEMAT Russian Chapter, e-mail: bpozin@ec-leasing.ru)

A New Software Engineering

The shortcomings of the existing state of software engineering, presents the main ideas and directions of "refounding" of software engineering as an engineering discipline. SEMAT kernel — a minimum set of entities of methodology or method (called alpha) and relations between them formulated. Defined set of characteristics of the states of alphas on the target software product creation with prescribed requirements to it. This provides an opportunity to systematically assess the "degree of progress" project and the "state of health" endeavor for its implementation. The ways of the phased development of software engineering as applied science are described.

Keywords: *SEMAT, SEMAT kernel, alpha, control states, method, practice, software system, area of concerns, control cards*

References

1. **Jacobson I., Ng P.-W., McMahon P. E., Spence I., Lidman S.** *The Essence of Software Engineering: Applying the SEMAT Kernel.* Addison Wesley, 2013. 224 p.

2. **Park J. S., Jacobson I., Myburgh B., Johnson P.** SEMAT vchera, segodnja i zavtra: perspektivy promyshlennogo ispol'zovanija. *Programmnaya ingeneria,* 2014, no. 11, pp. 6—16 (in Russian).

3. **Graziotin D., Abrahamsson P.** A Web-based modeling tool for the SEMAT Essence theory of software engineering. *Journal of Open Research Software*. 2013. vol. 1, no. 1: e4, available at: http://dx.doi.org/10.5334/jors.ad

4. **Jacobson I., Ng P.-W., McMahon P., Spence I., Lidman S.** The Essence of software engineering: the SEMAT kernel. *ACM Queue*, 2012, vol. 10, no. 10, available at: http://queue.acm.org/detail.cfm?id=2389616

5. **Object** Management Group. Essence—Kernel and language for software engineering methods. 2014, available at: http://www.omg.org/spec/Essence

6. **Jacobson I., Spence I., Ng P.-W.** Agile and SEMAT—Perfect Partners. *Communications of the ACM,* 2013, vol. 56, no. 11, pp. 53-59, available at: http://cacm.acm.org/magazines/2013/11/169027-agile-and-semat/abstract

7. **Kuhn T.** *The Structure of Scientific Revolutions*. Chicago: University of Chicago Press, 1962. 264 p.

УДК 004.423.4

В. А. Васенин, д-р физ.-мат. наук, зав. лабораторией, e-mail: vasenin@msu.ru,
М.А. Кривчиков, науч. сотр., e-mail: maxim.krivchikov@gmail.com,
НИИ механики МГУ им. М. В. Ломоносова

Формальные модели программ и языков программирования. Часть 1. Библиографический обзор 1930–1989 гг.

Статья содержит обзор и критический анализ научных публикаций с 1932 по 1989 г., тематика которых включает формальные модели вычислений, программ и языков программирования, в том числе работы по формальной верификации. Обзор выполнен в контексте истории появления и развития языков программирования, с учетом эволюции средств вычислительной техники и методов программной инженерии. Областью особого интереса является эволюция подходов к описанию вычислений в терминах различных разновидностей λ-исчисления.

Ключевые слова: *формальная верификация, языки программирования, предметно-ориентированные языки, формальная семантика программ, программная инженерия, библиография*

Введение

Методы формальной верификации — получения математических строгих доказательств выполнения программой поставленных перед ней требований — находятся в сфере активного интереса исследователей, начиная с 1960-х гг. В то же время математические теории, которые легли в основу таких методов, были построены существенно раньше, в рамках исследований в области математической логики и оснований математики. Так, первые публикации, которые рассматриваются в настоящей статье, относятся к началу 1930-х гг. и связаны с Проблемой разрешения Д. Гильберта. Дальнейшие исследования показали, что полнота некоторых моделей вычислений с позиций набора представимых алгоритмов в обязательном порядке влечет за собой неразрешимость нетривиальных свойств для представлений алгоритмов в таких моделях, а также противоречивость формальных логик, основанных на такой модели. По этой причине дальнейшие исследования в области математической логики были сосредоточены на формальных системах, менее мощных с позиций представления вычислений. В настоящее время такие системы применяют, в частности, и для решения задачи формальной верификации. На истории развития одной разновидности таких формальных систем, а именно типизированного λ-исчисления, и сделан акцент в настоящей работе.

Хотя первые языки программирования третьего поколения, предоставляющие средства абстракции от аппаратного обеспечения, появились более чем через пятнадцать лет после публикации результатов отмеченных выше фундаментальных исследований, в их основе, как и в основе большинства современных языков программирования, лежали эквивалентные машине Тьюринга модели вычислений. В настоящей статье рассматривается история появления важных с позиций формальной верификации моделей и методов, таких как формальные модели программ и языков программирования, а также последствия, которые оказал отмеченный выбор исходной модели вычислений для языков программирования на задачу формальной верификации.

1. Модели вычислений (1930-е гг.)

В 1930-е гг. были созданы первые экземпляры программируемых вычислительных машин. В частности, в 1936 г. К. Цузе была построена электромеханическая машина Z1 — предшественник одной из первых программируемых машин Z3 [1]. Вычислительные машины Цузе не получили широкого распространения. Однако в тот же исторический период в рамках фундаментальной математики активно велись исследования, связанные с Проблемой разрешения (*Entscheidungsproblem*), сформулированной Д. Гильбертом в 1928 г. [2] и частично решенной в работах А. Чёрча [3] и А. Тьюринга [4]. Именно эти исследования и положили начало формальной теории вычислений. Истоки Проблемы разрешения заключались в кризисе оснований математики и плане Гильберта. Основная формулировка, из которой была выделена проблема, состояла из следующих трех вопросов: является ли математика полной, непротиворечивой и разрешимой. Первая и вторая теоремы Гёделя о неполноте [5] дали ответ на первые два вопроса. Для того чтобы дать ответ на третий вопрос было необходимо, прежде всего, строго определить понятие разрешимости.

Первые с исторической точки зрения результаты, которые следует отметить в связи с задачами

формальной верификации и построения формальных моделей программ — это формальные модели вычислений. На этих результатах в той или иной степени основаны все последующие исследования в указанных областях. К таким результатам следует отнести машины Тьюринга и Поста, а также λ-исчисление Чёрча.

В 1936 г. в работе [4] А. Тьюринг предложил концепцию абстрактного вычислителя, который в дальнейшем получил название "машина Тьюринга" (в 1938 г. были опубликованы исправления к оригинальной статье [6]). Машину Тьюринга можно неформально описать как совокупность бесконечной в обе стороны ленты, разделенной на ячейки, в которых записаны символы определенного алфавита и управляющего устройства. Управляющее устройство имеет состояние, читающую и пишущую головку и набор правил перехода, задающих для входных состояния и символа, считанного с ленты, выходные состояние, символ, записываемый на ленту, и команду изменения позиции головки. Эмиль Пост в работе [7], вышедшей через пять месяцев после публикации первой работы Тьюринга, предложил более простой вариант такой машины, основные отличия которой от модели Тьюринга можно описать следующим образом: алфавит ограничивается двумя символами ("отмеченная" и "неотмеченная" ячейки), а правила перехода задаются с помощью фиксированного набора из шести инструкций.

Исходные результаты А. Чёрча были впервые опубликованы в 1932 г. как формальный язык, предлагаемый к использованию в качестве оснований математики [8]. Однако в 1935 г. Клини и Россером был сформулирован парадокс [9], демонстрирующий противоречивость результатов Чёрча. В 1936 г. Чёрч изолировал аспекты формальной системы, относящиеся к вычислимости в публикации [10]. Парадокс Клини—Россера и его упрощение в виде парадокса Карри для λ-исчисления позволяют установить важный факт. Его суть в том, что если некоторая модель вычислений допускает незавершимость, то она является противоречивой как логическая система. В 1940 г. Чёрч опубликовал работу [11], в которой рассматривалось λ-исчисление с простыми типами. В этой работе принципы λ-исчисления объединялись с теорией типов в стиле Principia mathematica Рассела и Уайтхеда. Для ограничения выразительной мощности системы к термам приписываются метки, называемые типами. Типы формируются согласно определенным правилам, а именно: задано множество базовых типов и комбинатор типов "$a \to b$", с использованием которого определяется тип функции между значениями типов a и b. Далее вводится отношение типизации, обеспечивающее соответствие между типом аргумента функции и типом правой части в операции приложения. Такое исчисление обладает рядом важных свойств, в числе которых сильная нормализация. Согласно свойству сильной нормализации, любой корректно типизированный терм исчисления за конечную последовательность редукций приводится к нормальной форме, единственной вне зависимости от порядка редукций. Важное свойство λ-исчисления представлено в теореме Чёрча—Россера. Согласно этой теореме, нетипизированное λ-исчисление конфлюэнтно, т. е. для любых двух термов, полученных из данного путем применения правил редукции к различным его подтермам, существует общий терм, достижимый из каждого результата с помощью применения правил редукции. Этот факт означает, что любой терм может иметь не более одной нормальной формы относительно правил редукции. Свойство Чёрча—Россера выполняется также для многих других разновидностей исчисления, в том числе для просто типизированного.

Отметим, что парадокс Клини—Россера был построен также для комбинаторной логики, открытой Шейнфинкелем, а затем, независимо — Карри. Комбинаторная логика также может служить моделью вычислений, эквивалентной машине Тьюринга. При этом она концептуально близка к λ-исчислению.

2. Появление языков программирования (1950-е гг.)

В течение пятого десятилетия XX века были разработаны и запущены первые широко используемые вычислительные машины, такие как ENIAC. Активно развивались исследования в области теории построения машин. В числе подобных результатов можно отметить результаты, представленные в статье Бёркса—Голдстайна—фон Неймана [12], определяющие основные принципы построения вычислительных устройств, которые используются и в настоящее время. В 1948—1951 гг. под руководством С. А. Лебедева была построена первая вычислительная машина в СССР — малая электронная счетная машина (МЭСМ). В течение 1950-х гг. как в Советском Союзе, так и за рубежом велась разработка новых версий вычислительных машин. В числе отечественных ЭВМ, выпущенных в 1950-х гг., можно отметить БЭСМ-1, первую серийную ЭВМ "Стрела" и серийный аналог БЭСМ-1 — БЭСМ-2.

В связи с изложенным выше появилась практическая необходимость в создании средств разработки программ для таких машин. Как результат работы на этом направлении в течение 1950-х гг. были разработаны первые языки программирования и их базовая теория. Программирование ENIAC представляло собой подключение кабелей и переключателей таким образом, чтобы выстроить вычислительные модули в систему, способную решить задачу. Перфокарты на этом этапе использовались только для ввода и вывода данных [13]. Для машин фон Неймановской архитектуры программирование проводилось уже с использованием машинного кода, т. е. числового представления команд. Впоследствие были разработаны так называемые "языки второго поколения" или "языки ассемблера", в которых числовое представление команд заменялось на мнемонические инструкции, отражающие специфику машины. Одной из первых машин, для которой

использовался язык ассемблера, была EDSAC [14], запущенная в 1949 г.

Дальнейшее развитие языков второго поколения привело к введению макрокоманд, т. е. мнемонических команд, которые разворачивались более чем в одну машинную инструкцию. Затем появились языки программирования третьего поколения, такие как Fortran (1957 г. [15]), LISP (1958 г. [16]), ALGOL (1958 г. [17]) и COBOL (1959 г., упоминается, в частности, в работе [18]). Необходимо отметить, что все перечисленные языки, за исключением языка ALGOL, который стал родоначальником ветви императивных языков программирования, развиваются и широко используются и в настоящее время. Язык Fortran в настоящее время остается одним из наиболее удобных средств для научных расчетов. Язык COBOL за 1960–1980-е гг. использовался для написания большого числа ориентированных на экономическое применение программ, которые до сих пор обрабатывают значительную часть финансовых транзакций в зарубежной экономике. Следует отметить, что ситуация с языком COBOL подчеркивает актуальность верификации в приложении к задаче реинжиниринга крупных программных комплексов. Программы на этом языке представляют собой яркий пример унаследованного кода, который разрабатывался в течение многих лет, в условиях постепенной смены команд специалистов. Язык COBOL используется до сих пор. Его вывод из эксплуатации чреват большими рисками, поскольку на настоящее время не существует методов реинжиниринга, предоставляющих строгие математические гарантии сохранения кодом важных свойств программного обеспечения и одновременно не требующих больших ресурсозатрат.

Программирование советских машин проводилось, как и в зарубежной практике, с использованием сначала машинных кодов, затем языков второго поколения "Автокод" и, наконец, проблемно-ориентированных языков третьего поколения [19].

Первый из фундаментальных результатов, который необходимо упомянуть в настоящем разделе, был получен в 1943 г. Первая формальная модель конечного автомата использовалась для моделирования нейросетей [20]. В дальнейшем теория автоматов активно развивалась, в том числе в приложении к решению задачи формальной верификации. Здесь следует отметить, что автоматные модели особенно удобны для верификации на модели (один из таких подходов — *model checking*, рассматривается в следующих разделах). В качестве распространенного на практике можно привести подход, в соответствии с которым можно перечислить все состояния и входы конечного автомата на конечном алфавите, что позволяет проводить верификацию достаточно простых моделей методом полного перебора. В 1956 г. Клини установил связь между конечными автоматами и регулярными выражениями [21] (оригинальный отчет об исследовании в рамках проекта RAND [22] датируется декабрем 1951 г.). В том же году Хомски опубликовал первые результаты по иерархии формальных грамматик [23].

Важным с точки зрения фундаментальных ограничений формальной верификации является результат Райса о неразрешимости проблемы эквивалентности вычислимых функций [24], опубликованный в 1953 г. Теорема Райса гласит, что для любого нетривиального свойства вычислимых функций алгоритмически неразрешима задача установления того факта, вычисляет ли произвольный алгоритм функцию с таким свойством. В соответствии с этой теоремой не может существовать универсальных автоматических методов верификации для нетривиальных свойств. Отметим, что под нетривиальным в теореме Райса понимается такое свойство, множество удовлетворяющих которому вычислимых функций не является пустым и не равно всему множеству вычислимых функций.

Отметим следующие две формальные модели вычислений, которые появились в шестом десятилетии XX века: модель частично-рекурсивных функций Клини [25] и нормальные алгорифмы Маркова [26]. Последняя модель оказала большое влияние на развитие систем переписывания термов, на ней был основан язык программирования Рефал. Основные идеи модели Маркова встречаются также в конструкциях сопоставления с образцом современных высокоуровневых языков программирования (таких как Wolfram Language или Haskell).

В 1952/1953 учебном году на кафедре вычислительной математики МГУ А. А. Ляпунов прочитал курс лекций "Принципы программирования", который стал первым курсом по программированию в СССР [27] (сокращенные материалы курса изданы в 1958 г. [28]). В рамках этого курса был предложен новый операторный метод описания программ, который далее был формализован Ю. И. Яновым [29] и получил название "схемы программы". Этот курс положил основы отечественной теории программирования. В числе основных задач этой теории Ляпунов выделил следующие две: автоматизация построения программ, т. е. построение программирующей программы, которая осуществляла бы трансляцию с операторной записи в машинные коды; оптимизация первоначально построенной программы, т. е. выделение среди множества эквивалентных программ оптимальной. В дальнейшем исследования в этой области, результаты которых будут рассмотрены в следующем разделе, были продолжены А. П. Ершовым.

С позиций инженерии программ, 1950-е гг. можно охарактеризовать тем, что практическая необходимость разработки программ повлекла за собой развитие теории программирования. Однако дефицит ресурсов (в том числе с точки зрения производительности и доступной памяти) в первых вычислительных машинах не позволял выполнять постановку крупных задач, для решения которых могли бы потребоваться методы проектирования и формализованные процессы разработки.

3. Развитие программирования (1960-е гг.)

Основной тенденцией развития вычислительной техники в 1960-х гг. было совершенствование элементной базы. Для построения ЭВМ стали использовать сначала полупроводниковые элементы, а затем и интегральные схемы, что позволило умень-

шить размер и увеличить производительность машин. К 1960-м гг. относится также появление специализированных бортовых ЭВМ, которые предназначались для обеспечения работоспособности авиации, космических аппаратов и систем ПВО и ПРО [30]. При разработке программного обеспечения для такого рода систем основным требованием являлось исполнение в режиме реального времени. Оно означает, что время реакции программной системы на внешние события должно быть ограничено сверху, причем абсолютное значение этого ограничения должно быть адекватно решаемой задаче. Это требование было новым для разработчиков. Для его выполнения было необходимо использовать другие методы, отличные от распространенных в то время методов решения расчетных задач, как правило, не ограниченных по времени.

Отметим некоторые важные результаты по формальным моделям, получившим в дальнейшем большое распространение в приложении к задаче верификации программного обеспечения. Первым таким результатом являются сети Петри, представленные в диссертации К. А. Петри [31]. Сети Петри представляют собой модель состояний и переходов, описываемую двудольным ориентированным графом, вершины которого называют позициями и переходами. В позициях могут быть размещены маркеры, которые перемещаются по сети при событии срабатывания переходов. Этот формализм широко используется для описания и верификации конкурентных и распределенных компьютерных систем. Вторым результатом являются автоматы Бюхи [32], первые представители класса ω-автоматов. Этот класс автоматов оперирует бесконечными входными строками и используется на практике для моделирования программ, не предусматривающих остановки, таких как операционные системы и системы управления.

С позиций настоящей работы одним из наиболее важных направлений исследований в 1960-е гг. было применение формальных методов к языкам программирования. В рамках разработки языка программирования Algol-58 Д. Бэкус предложил использование "металингвистических формул" для описания синтаксиса языка [17]. Этот формализм получил название нормальной формы Бэкуса (BNF). В дальнейшем, при разработке языка Algol-60 [33], формализм был упрощен П. Науром, после чего название формализма стало расшифровываться как "форма Бэкуса—Наура". На настоящее время этот формализм в совокупности с различными расширениями (EBNF, ABNF) является наиболее распространенным языком описания формальных грамматик языков программирования, одна из его версий (EBNF) принята ISO в качестве международного стандарта.

К концу 1960-х гг. в СССР и за рубежом вошел в употребление термин "программная инженерия", однако многие важные системные результаты в области инженерии программ появились только в следующем десятилетии.

Распространение языков программирования в 1960-х гг. привело к осознанию факта, который можно сформулировать следующим образом: непосредственной связи между программами, записанными в каких-либо языках программирования, и абстрактными моделями вычислений нет. Для того чтобы получить возможность доказательства свойств программ, а не алгоритмов, необходимо некоторое связующее звено между исходными кодами и математическими моделями. Вероятно, именно с этим фактом связано появление к концу 1960-х гг. трех основных подходов к описанию формальной семантики программ, а именно — операционной, аксиоматической и денотационной семантик.

Подход на основе операционной семантики описывает значение конструкций языка программирования с помощью непосредственного указания способа вычисления каждой конструкции в терминах абстрактного вычислителя. Впервые концепция операционной семантики была сформулирована и использована при разработке языка программирования Algol-68 [34].

Аксиоматический подход к описанию семантики программ был разработан Флойдом [35] и Хоаром [36] в 1967—1969 гг. Аксиоматическая семантика не определяет непосредственно значение, как некоторое заданное число, промежуточный или конечный результат вычисления отдельных конструкций или программы в целом. Вместо этого задается определенный набор свойств отдельных ее конструкций. Свойства конструкций описываются как аксиомы и правила вывода некоторой формальной системы, а свойства программы, в свою очередь, строятся как выводы из аксиом и правил вывода данной системы.

Денотационная семантика описывает значение программы с помощью специальных объектов, называемых денотациями. Основные положения этого подхода были разработаны Д. Скоттом и К. Стрейтчи в конце 1960-х гг. (первые материалы были опубликованы в 1970 г. [37]). Существенный вклад в теорию доменов, заложившую основу денотационной семантики, внес Ю. Л. Ершов в рамках серии работ по А-пространствам, первая из которых вышла в 1972 г. [38]. В роли денотаций могут выступать числа, функции или некоторые конструкции, такие как домены Скотта—Ершова. Денотационная семантика не определяет последовательность шагов, которые необходимы для вычисления выражений языка. С одной стороны, это является преимуществом, так как позволяет исследовать семантику программы как некоторый цельный объект, а не как последовательность шагов, что происходит в случае операционной семантики. С другой стороны, для императивных языков, в которых порядок выполнения действий имеет значение, это обстоятельство усложняет модель. Введение в методы описания денотационной семантики языков программирования изложено в работе [39].

В заключение настоящего раздела следует отметить работу [40], в которой предлагается использование формальной семантики языка программирования для автоматического построения транслятора записанных в рассматриваемом языке программ в машинный код.

4. Становление формальных моделей и методов программной инженерии (1970-е гг.)

В течение 1970-х гг. объемы производства вычислительной техники значительно выросли: от создания единичных, уникальных ЭВМ происходил переход к серийному производству моделей. Как в СССР, так и за рубежом серийно выпускались несколько семейств ЭВМ одновременно. Сфера задач, решение которых зависело от эффективного использования вычислительной техники, расширялась от научно-исследовательских и оборонных к информационным и экономическим. Сложность разрабатываемых в то время программных систем постоянно повышалась. В связи с этим именно в 1970-е гг. произошло становление отечественной программной инженерии как отдельной области знания.

В 1970-х гг. в МНИИПА были проведены исследования, обобщающие имеющийся опыт разработки программных продуктов в терминах экономической эффективности производства [19]. В результате исследований была подтверждена тенденция к повышению производительности труда программистов. Однако по сравнению с США имело место отставание как в объеме производства программных продуктов, так и в производительности труда. Кроме того, если в США относительная стоимость производства аппаратного и программного обеспечения оценивалась как 1:2—1:4 с перевесом и тенденцией к увеличению доли программного обеспечения, то в СССР аналогичные затраты рассчитывались как 2:1, что приводило к недооценке затрат на производство. Практическим примером такой недооценки стал транслятор с языка Алгол-60. В докладе на конференции "Построение программирующих программ на основе языка Алгол-60", проходившей 22—24 декабря 1960 г. в МГУ им. М. В. Ломоносова, разработчиками системы была заложена трудоемкость 15 человеко-лет и сложность — 15 000 команд. Фактическая трудоемкость составила свыше 30 человеко-лет, а сложность — 45 000 команд [41].

Одной из важнейших для программной инженерии разработок в 1970-е гг. была система автоматизации разработки программного обеспечения (САРПО) ЯУЗА-6, созданная под руководством В. В. Липаева и Л. А. Серебровского [19]. Эта система позволяла вести разработку программного обеспечения для ограниченных по характеристикам систем реального времени с использованием универсальных вычислительных машин для поддержки разработки. Она содержала в себе практически все элементы современных сред разработки. В ней, в частности, были применены идеи кросс-компиляции и параллельного использования нескольких языков программирования. Применялись также автоматизированные механизмы управления проектами, такие как расчет длительности реализации функциональных подпрограмм; контроль процесса разработки; выпуск и корректировка технической документации; хранение исходного кода.

За рубежом в рассматриваемый период также происходило активное формирование инженерии программ как отдельной области знания. Период с 1965 по 1985 г., со ссылкой на речь на награждении Э. Дейкстры Премией Тьюринга [42], получил название "кризиса программного обеспечения" (*software crisis*). Кризис выражался в том, что в условиях роста сложности программных продуктов значительная часть проектов выходила за рамки бюджета и сроков; программное обеспечение было неэффективным, низкокачественным и часто не удовлетворяло требованиям. Многие проекты вообще не доходили до завершения. Одним из канонических примеров сверхсложных программных систем была операционная система IBM OS/360, разработка которой заняла более 10 лет с использованием огромного количества ресурсов (более 1000 программистов) [43]. В процессе выхода из кризиса значительные ресурсы были брошены на разработку программных средств, методов и технологий построения сложных программных систем. Одной из наиболее известных монографий, в которой рассматривались актуальные на тот момент вопросы инженерии программ, стала книга Ф. Брукса "Мифический человеко-месяц" [43]. Эта книга до сих пор рекомендуется для подготовки специалистов в области разработки программного обеспечения.

Развитие предложенных в конце 1960-х гг. подходов к описанию семантики языков программирования происходило в нескольких направлениях, но наиболее значимые результаты были получены для описания семантики параллельных вычислений. В 1971 г. Г. Бекич предложил концепцию алгебры процессов, которая описывала семантику "квазипараллельного исполнения процессов" с использованием операторов последовательной и квазипараллельной композиции [44]. В рамках этой концепции исполнение параллельных процессов представлялось как недетерминированное слияние элементарных шагов (т. е. исполнение рассматривалось как последовательное с произвольным порядком). К порожденному этой концепцией классу можно отнести такие широко распространенные модели конкурентного исполнения, как CSP (*Communicating Sequential Processes*) Хоара [45] и CCS (*Calculus of Communication Systems*) Милнера [46]. В модели динамического параллельного исполнения программ используется аналогичный подход. Его суть в том, что семантика параллельно исполняющихся процессов строится как недетерминированное слияние последовательностей атомарных шагов.

К концу 1970-х гг. можно отнести также первые публикации, рассматривающие использование темпоральной логики для верификации программ, в том числе линейную темпоральную логику (LTL) [47]. В последующие годы, как будет описано в следующем разделе, подход к верификации на основе темпоральной логики будет использован как основа при разработке методов верификации на моделях (*model checking*), одного из основных и широко используемых в настоящее время подходов к верификации программ.

Для описания семантики рекурсивных программ было предложено использование наименьших неподвижных точек [48, 49]. С позиций аксиоматической семантики важным результатом стало использование

подхода слабейших предусловий (в рамках общего подхода преобразователей предикатов), предложенное Дейкстрой в 1975 г. [50]. Этот подход получил широкое применение в средствах статического анализа программ на предмет выполнения утверждений о состоянии в процессе их выполнения.

К 1970-м гг. относится появление важного расширения типизированного λ-исчисления. Это полиморфное λ-исчисление, называемое также типизированным λ-исчислением второго порядка (λ2), или, согласно исходному названию, *System F* (Система *F*). Основной особенностью этого расширения является возможность конструирования термов, зависящих от типов, т. е. абстрагирование от конкретных типов. Полиморфное λ-исчисление считается формализацией понятия параметрического полиморфизма в языках программирования, впервые рассмотренного К. Стрейтчи в конце 1960-х гг. (курс лекций был повторно опубликован в 2000 г. [51]). Эта система, наравне с ее расширением — полиморфным λ-исчислением с конструкторами типов (*System* F_ω), послужила основой для таких языков программирования, как Haskell, и семейства языков ML.

5. Развитие программной инженерии (1980-е гг.)

В течение 1980-х гг. в СССР в связи с решением контролирующих органов использовать в качестве образца для Единой Системы ЭВМ (ЕС ЭВМ) систему IBM System/360 оригинальное развитие вычислительной техники практически прекратилось. Исключением стали системы реального времени для военно-промышленного комплекса. Однако развитие программной инженерии продолжалось. Одним из наиболее существенных достижений отечественной программной инженерии была научно-исследовательская работа (НИР) "Прометей", которая проводилась с 1979 г. В выполнении этой работы принимали участие более 400 специалистов различных предприятий военно-промышленного комплекса. Задачей научно-исследовательской работы было проведение комплексного анализа и оптимизация процессов разработки крупных программных продуктов. В рамках этой работы были получены результаты, связанные со стадией тестирования и отладки, в том числе: методы автоматизированного тестирования и анализа уровня покрытия тестами исходного кода с использованием методов теории графов; применение технологий виртуализации в целях проведения отладки на универсальных машинах разработчиков; имитационное моделирование воздействий внешней среды. На основе указанных результатов была модифицирована упомянутая выше система ЯУЗА-6. Методы и средства поддержки технологических процессов, разработанные в рамках НИР "Прометей", использовались на многих предприятиях до конца 1990-х гг. [19].

Из числа работ, затрагивающих вопросы общих подходов к программной инженерии, следует отметить статью П. Наура [52], согласно которой разработку программного обеспечения следует рассматривать в первую очередь как непрерывный процесс построения и модификации теории (знания) о том, каким образом процесс выполнения программы позволяет получить решение практических задач. В том числе такая модель адекватно описывает отмеченные в 1970—1980-е гг. сложности. Во-первых, при увеличении команды разработчиков общая производительность, как правило, не повышается, а в случае больших команд — снижается. Во-вторых, как отмечает Наур, на практике существуют значительные сложности, которые появляются при организации деятельности новой команды разработчиков над унаследованным кодом. Первый случай может объясняться процессами передачи знаний о ходе решения задач программой и, что характерно для больших команд, существенной фрагментацией таких знаний. Состояние фрагментации заключается в том, что каждый отдельный член команды является носителем эксклюзивных знаний о части системы, за которую он отвечает. Нетривиальные модификации программы при этом требуют согласованного применения таких знаний о различных частях программы. Во втором случае утверждается об утере таких знаний и необходимости их восстановления новой командой по исходному коду и документации.

К 1980-м гг. относятся первые редакции международных стандартов, связанных с программной инженерией, такие как сборник *Software Engineering Standards* [53]. Из числа нормативных документов необходимо выделить также серию стандартов ISO 9000, которая определяет требования к системам контроля качества на производстве в целом. Положения этой серии стандартов применимы и к технологическим процессам разработки программных продуктов, в частности, современная интерпретация стандартов ISO 9000 с позиций программной инженерии содержится в стандарте ISO/IEC 90003 [54].

На фоне продолжающегося "кризиса программного обеспечения", в связи с активной разработкой новых методов, инструментальных средств и стандартов программной инженерии, новые результаты воспринимались индустрией как универсальные средства, способствующие разрешению в кратчайшие сроки всех проблем в области разработки программных продуктов. Однако на практике после внедрения таких результатов в процесс разработки ожидаемого эффекта не наблюдалось и ажиотажный интерес к новым средствам спадал. В 1987 г. Ф. Брукс в работе [55] утверждал, что "не существует таких технологий или методик управления проектами, с помощью которых можно добиться роста на один порядок за 10 лет в показателях производительности, надежности и простоты программных систем".

Отношение, аналогичное общей ситуации с новыми подходами, складывалось в 1980-х гг. и к рассматриваемым в настоящей работе формальным методам программной инженерии. Изначально они были восприняты как многообещающие средства, позволяющие гарантировать корректность программных про-

дуктов. Однако сложность использования, большое количество недостаточно глубоко проработанных задач и высокие требования к квалификации способных использовать их специалистов привели к разочарованию в этих методах [56].

С конца 1980-х гг. верификация на модели [57] стала одним из наиболее распространенных подходов к формальной верификации программного обеспечения. Поскольку в настоящей работе рассматривается другой подход, а именно дедуктивная верификация и программирование с зависимыми типами, работы по верификации на модели не рассматриваются. В качестве дальнейших источников по этому подходу может быть рекомендована книга "Principles of Model Checking" [58], а также труды ежегодных международных конференций "Verification, Model Checking, and Abstract Interpretation" и симпозиумов "Model Checking Software".

К концу 1980-х гг. следует отнести основные теоретические результаты, связанные с полиморфным λ-исчислением второго порядка с зависимыми типами (исчислением конструкций, *Calculus of Constructions*). В первую очередь следует отметить работы Т. Кокана. В работе [59] рассматривается парадокс Жирара применительно к таким исчислениям. В качестве одного из способов преодоления парадокса предложено использование иерархии универсумов. В работе [60] впервые представлено доказательство строгой нормализации для определенного в ней же исчисления конструкций. Отчет INRIA [61] содержит предложения по дальнейшей работе над формальной системой, в том числе по введению индуктивных типов. Из числа других исследователей следует отметить Чж. Луо, которым опубликована разновидность исчисления конструкций ECC [62], дополненная конструктором типов зависимых сумм.

В работе [63] рассмотрено описание модулей (в реализации, используемой в том числе языками семейства ML) с позиций зависимых типов. Результаты аналогичного подхода для языка SOL на базе λ-исчисления второго порядка представлены в работе [64]. В курсе лекций [65], прочитанном П. Мартин-Лёфом в 1980 г., рассматривается интуиционистская теория типов, которая использует запись, аналогичную λ-исчислению с зависимыми типами для описания логического исчисления. Таким образом, интуиционистскую теорию типов можно рассматривать как логическую интерпретацию λ-исчисления с зависимыми типами. В качестве введения в теорию типов может быть рекомендована работа [66].

Работа [67] посвящена извлечению программ исчисления F_ω из термов-доказательств исчисления конструкций. Внимание уделяется, в частности, вопросам удаления нерелевантной с позиций вычислений информации, такой как типы или доказательства информативных утверждений. Для последнего используется специальный тип Prop, который присваивается всем информативным утверждениям и позволяет в процессе извлечения отличать их от релевантных с позиций вычислений функций, имеющих тип Set.

Практический аспект использования λ-исчисления с зависимыми типами, а именно реализация процедуры частичного разрешения частного случая формул (*unification problem*) в разновидности исчисления без полиморфизма и типов высших порядков, рассматривается в работе [68]. Такая процедура используется, например, для автоматизированного доказательства утверждений и, в более общем случае, для сопоставления с образцом в языках программирования.

Связь различных разновидностей λ-исчисления, в том числе исчисления с зависимыми типами, математической логики, теории категорий и денотационной семантики является предметом анализа в статье [69]. Статья содержит обширную библиографию, включающую большое число публикаций в предметной области за 1970—1988 гг.

Один из возможных алгоритмов проверки типов для исчисления конструкций представлен в работе [70]. Этот алгоритм был разработан для среды автоматизированных доказательств LEGO. Существенной особенностью этого алгоритма является шаг проверки ограничений на уровни типов — аналогичные подходы используются при реализации алгоритмов проверки выводов в рамках теории типов и в настоящее время.

Заключение

В настоящей статье рассмотрены исторические публикации, начиная от первых формальных моделей вычислений 1930-х гг. и заканчивая, возможно, одним из центральных для методов формальной верификации делением на теоретико-модельные и теоретико-доказательные методы формальной верификации. В рамках теоретико-модельных методов рассматриваются модели программ с конечно описываемым множеством состояний, чаще всего — автоматные модели. Для таких моделей возможна автоматическая верификация, однако приведение программ к таким моделям в общем случае является отдельной, весьма нетривиальной задачей. В рамках теоретико-доказательных (дедуктивных) методов доказательства свойств программ строятся вручную, с частичной автоматизацией, как выводы в некоторой логике, как правило, логике высшего порядка. На настоящее время известны результаты успешного применения к задачам формальной верификации методов каждого из классов. Тем не менее наиболее сложными вопросами с позиций формальной верификации остается представление программ и их спецификаций в виде, допускающем использование тех или иных методов.

Хотя основные подходы к описанию формальной семантики языков программирования сформировались к началу 1970-х гг., применение таких подходов к существующим распространенным языкам программирования общего назначения на практике и в настоящее время остается сложной задачей. Наличие формальной семантики не требуется, в частности, международными комитетами по стандартизации (такими как ISO/IEC JTC1 или ECMA) при утверждении

стандарта языка, а исследования в области построения формальной семантики существующих языков носят в первую очередь инициативный характер.

Во второй части этого цикла статей предполагается рассмотреть современное состояние исследований в первую очередь с позиций приложения моделей типизированного λ-исчисления к задаче формальной верификации. Будет рассмотрен также подход к разработке программного обеспечения на основе языково-ориентированного программирования. Применение этого подхода к задаче разработки верифицируемого программного обеспечения, на взгляд авторов, является одним из перспективных направлений дальнейших исследований.

Список литературы

1. **Rojas R.** Konrad Zuse's legacy: the architecture of the Z1 and Z3 // IEEE Annals of the History of Computing. 1997. Vol. 19, N. 2. P. 5—16.
2. **Hilbert D., Ackermann W.** Grundzüge der theoretischen Logik. Berlin: J. Springer, 1928.
3. **Church A.** An Unsolvable Problem of Elementary Number Theory // American Journal of Mathematics. 1936. Vol. 58, N. 2. P. 345—363.
4. **Turing A. M.** On computable numbers, with an application to the Entscheidungsproblem // J. of Math. 1936. Vol. 38, N. 1931. P. 173—198.
5. **Gödel K.** Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I // Monatshefte für Mathematik und Physik. 1931. Vol. 38, N. 1. P. 173—198.
6. **Turing A. M.** On Computable Numbers, with an Application to the Entscheidungsproblem. A Correction // Proceedings of the London Mathematical Society. 1938. S. 2. Vol. 43, N. 6. P. 544—546.
7. **Post E. L.** Finite combinatory processes — formulation 1 // The Journal of Symbolic Logic. 1936. Vol. 1. P. 103—105.
8. **Church A.** A Set of Postulates for the Foundation of Logic // The Annals of Mathematics. 1932. Vol. 33, N. 2. P. 346—366.
9. **Kleene S. C., Rosser J. B.** The Inconsistency of Certain Formal Logics // The Annals of Mathematics. 1935. Vol. 36, N. 3. P. 630—636.
10. **Church A.** A note on the Entscheidungsproblem // Journal of Symbolic Logic. 1936. N. 1. P. 40—41.
11. **Church A.** A formulation of the simple theory of types // The Journal of Symbolic Logic. 1940. Vol. 5, N. 02. P. 56—68.
12. **Neumann J. v., Goldstine H. H., Burks A. W.** Preliminary discussion of the logical design of an electronic computing instrument: Rep. Institute of Advanced Study. Princeton, NJ: 1946.
13. **Burks A. W., Burks A. R.** The ENIAC: The First General-Purpose Electronic Computer // Annals of the History of Computing. 1981. Vol. 3, N. 4. P. 310—389.
14. **Salomon D.** Assemblers And Loaders. Chichester, West Sussex, UK: Ellis Horwood Ltd, 1992.
15. **Backus J. W., Stern H., Ziller I. et al.** The FORTRAN automatic coding system // Western joint computer conference: Techniques for reliability on — IRE-AIEE-ACM'57 (Western). 26—28 February, 1957. ACM Press, 1957. P. 188—198.
16. **McCarthy J.** Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I // Communications of the ACM. 1960. Vol. 3, N. 4. P. 184—195.
17. **Backus J. W.** The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference // International Conference on Information Processing. UNESCO, 1959. P. 125—132.
18. **McCarthy J.** A Basis for a Mathematical Theory of Computation // Studies in Logic and the Foundations of Mathematics / Eds. by P. B. Hirschberg. Vol. 26 of Computer Programming and Formal Systems. Elsevier, 1959. P. 33—70.
19. **Липаев В. В.** Отечественная программная инженерия: фрагменты истории и проблемы. М.: СИНТЕГ, 2007. 312 с.
20. **McCulloch W. S., Pitts W.** A logical calculus of the ideas immanent in nervous activity // The Bulletin of Mathematical Biophysics. 1943. Vol. 5, N. 4. P. 115—133.
21. **Kleene S. C.** Representation of events in nerve nets and finite automata // Automata Studies / Eds. by C. Shannon, J. McCarthy. Princeton, NJ: Princeton University Press, 1956. P. 3—41.
22. **Representation** of events in nerve nets and finite automata: Rep. RAND Corporation: S. C. Kleene. Santa Monica, CA, USA: 1951. 98 p.
23. **Chomsky N.** Three models for the description of language // IEEE Transactions on Information Theory. 1956. Vol. 2, N. 3. P. 113—124.
24. **Rice H. G.** Classes of recursively enumerable sets and their decision problems // Transactions of the American Mathematical Society. 1953. Vol. 74, N. 2. P. 358—366.
25. **Kleene S. C.** Introduction to Metamathematics. New York: Van Nostrand, 1952.
26. **Марков А. А.** Теория алгорифмов. Труды математического института имени В. А. Стеклова № XLII. М.—Л.: Изд-во АН СССР, 1954.
27. **Подловченко Р. И., Ляпунов А. А.** А. П. Ершов в теории схем программ и развитие ее логических концепций. Андрей Петрович Ершов — ученый и человек / Под ред. А. Г. Марчук. Новосибирск: Издательство СО РАН, 2006.
28. **Янов Ю. И.** О логических схемах алгоритмов // Проблемы кибернетики: сборник статей. 1958. № 1. С. 75—127.
29. **Ляпунов А. А.** О логических схемах программ // Проблемы кибернетики: сборник статей. 1958. № 1. С 46—74.
30. **ECMA** International. Standard ECMA-335 — Common Language Infrastructure (CLI). 4 edition. Geneva, Switzerland, 2006. June.
31. **Petri C. A.** Kommunikation mit Automaten: Ph. D. thesis. Institut für instrumentelle Mathematik. Bonn, 1962.
32. **Buechi J. R.** On a Decision Method in Restricted Second-Order Arithmetic // International Congress on Logic, Methodology, and Philosophy of Science. Stanford University Press, 1962. P. 1—11.
33. **Revised** Report on the Algorithmic Language ALGOL 60 // Communications of the ACM / Eds. by P. Naur. 1963. Vol. 6, N. 1. P. 1—17.
34. **Lindsey C. H.** A history of ALGOL 68 // History of Programming Languages II / Eds. by T. J. Bergin, R. G. Gibson. New York, NY, USA: ACM, 1996. P. 27—96.
35. **Floyd R. W.** Assigning Meanings to Programs // Mathematical Aspects of Computer Science / Eds. by J. T. Schwartz. V. 19 of Proceedings of Symposia in Applied Mathematics. American Mathematical Society, 1967. P. 19—32.
36. **Hoare C. A. R.** An axiomatic basis for computer programming // Communications of the ACM. 1969. Vol. 12, N. 10. P. 576—580.
37. **Outline** of a Mathematical Theory of Computation: Rep. PRG02. Oxford University Computing Laboratory: D. S. Scott. Oxford, England: 1970. 30 p.
38. **Ершов Ю. Л.** Непрерывные решетки и А-пространства // Доклады Академии Наук СССР. 1972. Т. 207, № 3. С. 523—526.
39. **Schmidt D. A.** A Methodology for Language Development. Boston: Allyn and Bacon, 1986.
40. **Feldman J. A.** A Formal Semantics for Computer Languages and Its Application in a Compiler-compiler // Communications of the ACM. 1966. Vol. 9, N. 1. P. 3—9.
41. **Становление** программирования в СССР (переход ко второму поколению языков и машин). Препринт: 12. ВЦ СО АН СССР: Ершов А. П., Шура-Бура М. Р. 1976.
42. **Dijkstra E. W.** The humble programmer // Communications of the ACM. 1972. Vol. 15, N. 10. P. 859—866.
43. **Brooks F. P.** The Mythical Man-Month. Addison-Wesley, 1975.
44. **Towards** a mathematical theory of processes: Rep. TR 25.125. IBM Laboratory Vienna: H. Bekić. Vienna, Austria, 1971.
45. **Hoare C. A. R.** Communicating sequential processes // Communications of the ACM. 1978. Vol. 21, N. 8. P. 666—677.
46. **Milner R.** A calculus of communicating systems. Secaucus, NJ, USA: Springer-Verlag New York, Inc, 1982.
47. **Pnueli A.** The temporal logic of programs // 18th Annual Symposium on Foundations of Computer Science (SFCS 1977). IEEE, 1977. P. 46—57.
48. **De Bakker J. W., De Roeve W.** A calculus for recursive program schemes // 1st International Colloquium on Automata, Lanugages and Programming. 1972. P. 167—196.
49. **Park D. M. R.** Fixpoint induction and proof of program semantics // Machine Intelligence. 1970. N. 5. P. 59—78.
50. **Dijkstra E. W.** Guarded commands, non-determinacy and formal derivation of programs // Communications of the ACM. 1975. Vol. 18, N. 8. P. 453—457.
51. **Strachey C.** Fundamental concepts in programming languages // Higher-order and symbolic computation. 2000. Vol. 1, N. 1—2. P. 11—49.
52. **Naur P.** Programming as theory building // Microprocessing and Microprogramming. 1985. Vol. 15, N. 5. P. 253—261.

53. **IEEE** Standard Taxonomy for Software Engineering Standards. IEEE Std. 1002-1987. 1987.
54. **Software** Engineering — Guidelines for the application of ISO 9001:2008 to computer software. ISO/IEC Std. 90003:2014. — 2014.
55. **Brooks F. P.** No Silver Bullet // Computer. 1987. Vol. 20, N. 4. P. 10—19.
56. **Stevenson D. E.** 1001 reasons for not proving programs correct: A survey. Computer Science Dept, Clemson University. Citeseer, 1990.
57. **Clarke E. M., Grumberg O., Peled D. A.** Model Checking. Cambridge, Mass: The MIT Press, 1999.
58. **Baier C., Katoen J.-P.** Principles of Model Checking. MIT Press, 2008. 984 p.
59. **Coquand T.** An Analysis of Girard's Paradox // In Symposium on Logic in Computer Science. IEEE Computer Society Press, 1986. P. 227—236.
60. **Coquand T., Huet G.** The calculus of constructions // Information and Computation. 1988. Vol. 76, N. 2—3. P. 95—120.
61. **Metamathematical** investigations of a calculus of constructions: Rep. INRIA: T. Coquand. Rocquencourt, France, 1989. 32 p.
62. **Luo Z.** ECC, an extended calculus of constructions // Proceedings. Fourth Annual Symposium on Logic in Computer Science. IEEE Comput. Soc. Press, 1989. P. 386—395.
63. **MacQueen D. B.** Using dependent types to express modular structure // Proceedings of the 13th ACM SIGACT-SIGPLAN symposium on Principles of programming languages — POPL'86. ACM Press, 1986. P. 277—286.
64. **Mitchell J. C., Plotkin G. D.** Abstract types have existential type // ACM Transactions on Programming Languages and Systems. 1988. Vol. 10, N. 3. P. 470—502.
65. **Martin-Löf P.** Intuitionistic type theory / Eds. by G. Sambin. Napoli: Bibliopolis, 1984.
66. **Thompson S.** Type theory and functional programming. Addison—Wesley, 1991.
67. **Paulin-Mohring C.** Extracting Fomega's programs from proofs in the calculus of constructions // Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages — POPL'89. ACM Press, 1989. P. 89—104.
68. **Elliott C. M.** Higher-order Unification with Dependent Function Types // Proceedings of the 3rd international conference on Rewriting Techniques and Applications. New York: Springer-Verlag, 1989. P. 121—136.
69. **Longo G.** On church's formal theory of functions and functionals // Annals of Pure and Applied Logic. 1988. Vol. 40, N. 2. P. 93—133.
70. **Pollack R., Harper R.** Type Checking, Universe Polymorphism and Typical Ambiguity in the Calculus of Constructions // Proceedings of the International Joint Conference on Theory and Practice of Software Development. Barcelona, Spain, March 13—17. Berlin Heidelberg: Springer, 1989. P. 241—256.

V. A. Vasenin, Head of the Laboratory, e-mail: vasenin@msu.ru, **M. A. Krivchikov,** Researcher, e-mail: maxim.krivchikov@gmail.com, Institute of Mechanics, Lomonosov Moscow State University

Formal Models of Programming Languages and Programs. Part 1. Literature Review: 1930–1989

The main contribution of the present article is a historical literature review and a critical analysis of the publications on the following topics: the formal verification, formal models of software and the development of formally verifiable software. Publications on the listed topics are presented in a context of the software engineering development in general. The cited papers were published in the time frame of 1930–1989. One of the specific topic of interest of the presented research is the dependently-typed programming development from the untyped lambda-calculus to the calculus of constructions. The complexity introduced into general purpose programming languages from the formal verification point of view by the means of computational models having an inconsistent logical interpretation is noted. Fundamental results associated with the formal verification are also mentioned. In the last part of the article the commonly used distinction of the model-theoretic and the proof-theoretic (deductive) approaches to formal verification is considered.

As a conclusion, two points are stated. In the first place, the representation of the specification and especially the program code in a mathematically rigorous way nowadays is probably the most complex part of the formal verification problem. In the second place, despite active research since late 1960s, construction of the formal semantics for a general-purpose programming language is still a complex task. It is possible that for this reason the formal descrption of the programming language is still not required by international standards organizations and commitees.

Keywords: *formal verication, programming languages, domain-specific languages, programming language formal semantics, software engineering, bibliography*

References

1. **Rojas R.** Konrad Zuse's legacy: the architecture of the Z1 and Z3. *IEEE Annals of the History of Computing,* 1997, vol. 19, no. 2, pp. 5—16.
2. **Hilbert D., Ackermann W.** *Grundzuge der theoretischen Logik*. Berlin: J. Springer, 1928.
3. **Church A.** An Unsolvable Problem of Elementary Number Theory. *American Journal of Mathematics*, 1936. April, vol. 58, no. 2, pp. 345—363.
4. **Turing A. M.** On computable numbers, with an application to the Entscheidungsproblem. *J. of Math*., 1936, vol. 38, no. 1931, pp. 173—198.
5. **Godel K.** Uber formal unentscheidbare Satze der Principia Mathematica und verwandter Systeme I. *Monatshefte fur Mathematik und Physik*, 1931, vol. 38—38, no. 1, pp. 173—198.
6. **Turing A. M.** On Computable Numbers, with an Application to the Entscheidungsproblem. A Correction. *Proceedings of the London Mathematical Society.* 1938, S. 2. vol. 43, no. 6, pp. 544—546.
7. **Post E. L.** Finite combinatory processes — formulation 1. *The Journal of Symbolic Logic*, 1936, vol. 1, pp. 103—105.
8. **Church A.** A Set of Postulates for the Foundation of Logic. *The Annals of Mathematics*, 1932, vol. 33, no. 2, pp. 346—366.
9. **Kleene S. C., Rosser J. B.** The Inconsistency of Certain Formal Logics. *The Annals of Mathematics*, 1935, vol. 36, no. 3, pp. 630—630.
10. **Church A.** A note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1936, no. 1, pp. 40—41.
11. **Church A.** A formulation of the simple theory of types. *The Journal of Symbolic Logic.* 1940, vol. 5, no. 02, pp. 56—68.
12. **Neumann J. v., Goldstine H. H., Burks A. W.** Preliminary discussion of the logical design of an electronic computing instrument: Rep. Institute of Advanced Study. Princeton, NJ: 1946.

13. **Burks A. W., Burks A. R.** The ENIAC: The First General-Purpose Electronic Computer. *Annals of the History of Computing,* 1981, vol. 3, no. 4, pp. 310—389.
14. **Salomon D.** *Assemblers And Loaders.* Chichester, West Sussex, UK: Ellis Horwood Ltd, 1992.
15. **Backus J. W., Stern H., Ziller I. et al.** *The FORTRAN automatic coding system*. Western joint computer conference: Techniques for reliability on — IRE-AIEE-ACM'57 (Western). 26—28 February, 1957. ACM Press, 1957, pp. 188—198.
16. **McCarthy J.** Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I. *Communications of the ACM,* 1960, vol. 3, no. 4, pp. 184—195.
17. **Backus J. W.** The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference. International Conference on Information Processing. UNESCO, 1959, pp. 125—132.
18. **McCarthy J.** A Basis for a Mathematical Theory of Computation. Studies in Logic and the Foundations of Mathematics / Eds. by P. B. Hirschberg, vol. 26 of *Computer Programming and Formal Systems*. Elsevier, 1959, pp. 33—70.
19. **Lipaev V. V.** *Otechestvennaya programmnaya inzheneriya: fragmenti istorii i problemi*. M.: SINTEG, 2007. 312 p. (in Russia).
20. **McCulloch W. S., Pitts W.** A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*. 1943, vol. 5, no. 4, pp. 115—133.
21. **Kleene S. C.** Representation of events in nerve nets and finite automata. Automata Studies / Eds. by C. Shannon, J. McCarthy. Princeton, NJ: Princeton University Press, 1956, pp. 3—41.
22. **Representation** of events in nerve nets and finite automata: Rep. RAND Corporation: S. C. Kleene. Santa Monica, CA, USA: 1951. 98 p.
23. **Chomsky N.** Three models for the description of language. *IEEE Transactions on Information Theory*, 1956, vol. 2, no. 3, pp. 113—124.
24. **Rice H. G.** Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 1953, vol. 74, no. 2, pp. 358—366.
25. **Kleene S. C.** *Introduction to Metamathematics*. New York: Van Nostrand, 1952.
26. **Markov A. A.** *Teoriya algorithmov. Trudi matematicheskigo instituta imeni V. A. Steklova, N. XLII.* — Moskva, Leningrad: Izd-vo AN SSSR, 1954 (in Russian).
27. **Podlovchenko R. I., Lyapunov A. A.** A, pp. Ershov v teorii skhem programm i razvitie eyo logicheskih koncepciy. *Andrey Petrovich Ershov — uchenii i chelovek* / Eds. by. A. G. Marchuk. Novosibirsk: Izdatelstvo SO RAN, 2006 (in Russian).
28. **Janov Ju. I.** O logicheskih shemah algoritmov. *Problemy kibernetiki: sbornik statej,* 1958, no. 1. pp. 75—127 (in Russian).
29. **Lyapunov A. A.** O logicheskih shemah programm. *Problemy kibernetiki: sbornik statej*, 1958. no. 1. pp. 46—74 (in Russian).
30. **ECMA** International. Standard ECMA-335 — Common Language Infrastructure (CLI). 4 edition. Geneva, Switzerland, 2006. June.
31. **Petri C. A.** Kommunikation mit Automaten: Ph. D. thesis. Institut fur instrumentelle Mathematik. Bonn, 1962.
32. **Buechi J. R.** On a Decision Method in Restricted Second-Order Arithmetic. *International Congress on Logic, Methodology, and Philosophy of Science*. Stanford University Press, 1962, pp. 1—11.
33. **Revised** Report on the Algorithmic Language ALGOL 60. *Communications of the ACM* / Eds. by P. Naur. 1963, vol. 6, no. 1, pp. 1—17.
34. **Lindsey C. H.** A history of ALGOL 68. *History of Programming Languages II* / Eds. by T. J. Bergin, R. G. Gibson. New York, NY, USA: ACM, 1996, pp. 27—96.
35. **Floyd R. W.** Assigning Meanings to Programs. *Mathematical Aspects of Computer Science* / Eds. by J. T. Schwartz. *V. 19 of Proceedings of Symposia in Applied Mathematics*. American Mathematical Society, 1967, pp. 19—32.
36. **Hoare C. A. R.** An axiomatic basis for computer programming. *Communications of the ACM*, 1969, vol. 12, no. 10, pp. 576—580.
37. **Outline** of a Mathematical Theory of Computation: Rep. PRG02. Oxford University Computing Laboratory: D. S. Scott. Oxford, England: 1970. 30 p.
38. **Ershov Ju. L.** Nepreryvnye reshetki i A-prostranstva. *Doklady Akademii Nauk SSSR*, 1972, vol. 207, no. 3. pp. 523—526 (in Russian).
39. **Schmidt D. A.** *A Methodology for Language Development*. Boston: Allyn and Bacon, 1986.
40. **Feldman J. A.** A Formal Semantics for Computer Languages and Its Application in a Compiler-compiler. *Communications of the ACM*, 1966, vol. 9, no. 1, pp. 3—9.
41. **Stanovlenie** programmirovanija v SSSR (perehod ko vtoromu pokoleniju jazykov i mashin): Preprint: 12 / VC SO AN SSSR: A. P. Ershov, M. R. Shura-Bura, 1976 (in Russian).
42. **Dijkstra E. W.** The humble programmer. *Communications of the ACM*, 1972, vol. 15, no. 10, pp. 859—866.
43. **Brooks F. P.** *The Mythical Man-Month*. Addison-Wesley, 1975.
44. **Towards** a mathematical theory of processes: Rep. TR 25.125. IBM Laboratory Vienna: H. Bekić. Vienna, Austria, 1971.
45. **Hoare C. A. R.** Communicating sequential processes. *Communications of the ACM*, 1978, vol. 21, no. 8, pp. 666—677.
46. **Milner R.** *A calculus of communicating systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc, 1982.
47. **Pnueli A.** The temporal logic of programs. *18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*. IEEE, 1977, pp. 46—57.
48. **De Bakker J. W., De Roeve W.** A calculus for recursive program schemes. *Ist International Colloquium on Automata, Lanugages and Programming,* 1972, pp. 167—196.
49. **Park D. M. R.** Fixpoint induction and proof of program semantics. *Machine Intelligence,* 1970. no. 5, pp. 59—78.
50. **Dijkstra E. W.** Guarded commands, non-determinacy and formal derivation of programs. *Communications of the ACM,* 1975, vol. 18, no. 8, pp. 453—457.
51. **Strachey C.** Fundamental concepts in programming languages. *Higher-order and symbolic computation*. 2000, vol. 1, no. 1—2, pp. 11—49.
52. **Naur P.** Programming as theory building. *Microprocessing and Microprogramming*, 1985, vol. 15, no. 5, pp. 253—261.
53. **IEEE** Standard Taxonomy for Software Engineering Standards. IEEE Std. 1002-1987. 1987.
54. **Software** Engineering — Guidelines for the application of ISO 9001:2008 to computer software. ISO/IEC Std. 90003:2014. 2014.
55. **Brooks F. P.** No Silver Bullet. *Computer*, 1987, vol. 20, no. 4, pp. 10—19.
56. **Stevenson D. E.** 1001 reasons for not proving programs correct: A survey. Computer Science Dept, Clemson University. Citeseer, 1990.
57. **Clarke E. M., Grumberg O., Peled D. A.** *Model Checking*. Cambridge, Mass: The MIT Press, 1999.
58. **Baier C., Katoen J.-P.** *Principles of Model Checking*. MIT Press, 2008. 984 p.
59. **Coquand T.** An Analysis of Girard's Paradox. *In Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1986, pp. 227—236.
60. **Coquand T., Huet G.** The calculus of constructions // *Information and Computation*, 1988, vol. 76, no. 2—3, pp. 95—120.
61. **Metamathematical** investigations of a calculus of constructions: Rep. INRIA: T. Coquand. Rocquencourt, France, 1989, pp. 32.
62. **Luo Z.** ECC, an extended calculus of constructions. *Proceedings. Fourth Annual Symposium on Logic in Computer Science*. IEEE Comput. Soc. Press, 1989, pp. 386—395.
63. **MacQueen D. B.** *Using dependent types to express modular structure. Proceedings of the 13th ACM SIGACT-SIGPLAN symposium on Principles of programming languages — POPL'86*. ACM Press, 1986, pp. 277—286.
64. **Mitchell J. C., Plotkin G. D.** Abstract types have existential type. *ACM Transactions on Programming Languages and Systems*, 1988, vol. 10, no. 3, pp. 470—502.
65. **Martin-Löf P.** *Intuitionistic type theory* / Eds. by G. Sambin. Napoli: Bibliopolis, 1984.
66. **Thompson S.** *Type theory and functional programming*. Addison—Wesley, 1991.
67. **Paulin-Mohring C.** Extracting Fomega's programs from proofs in the calculus of constructions. *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages — POPL'89*. ACM Press, 1989, pp. 89—104.
68. **Elliott C. M.** Higher-order Unification with Dependent Function Types. *Proceedings of the 3rd international conference on Rewriting Techniques and Applications*. New York: Springer-Verlag, 1989, pp. 121—136.
69. **Longo G.** On church's formal theory of functions and functionals. *Annals of Pure and Applied Logic*, 1988, vol. 40, no. 2, pp. 93—133.
70. **Pollack R., Harper R.** Type Checking, Universe Polymorphism and Typical Ambiguity in the Calculus of Constructions. *Proceedings of the International Joint Conference on Theory and Practice of Software Development*. Barcelona, Spain, March 13—17. Berlin Heidelberg: Springer, 1989, pp. 241—256.

УДК 004.4

Д. Е. Намиот, канд. физ.-мат. наук, ст. науч. сотр., e-mail: dnamiot@gmail.com,
Московский государственный университет им. М.В. Ломоносова

Метаданные в REST-модели

Проанализированы подходы к представлению метаданных в программных моделях, основанных на Representational State Transfer (REST) подходе. В настоящее время архитектура на основе REST — одна из наиболее часто используемых программных архитектур. Распространение данного подхода, не в последнюю очередь было связано с простотой программной реализации и лаконичностью его описания в сравнении с сервис-ориентированной архитектурой (Service Oriented Architecture), с которой REST чаще всего и сравнивают. Вместе с тем стремление к простоте (лаконичности) реализации привело к тому, что такой элемент, как метаданные, в классической модели REST отсутствует. Настоящая работа посвящена поиску ответов на вопросы: зачем метаданные могут быть нужны; что и как они могут представлять; какие инструментальные средства могут быть использованы для их представления в системах на основе REST-архитектуры.

Ключевые слова: *веб-сервисы, SOAP, REST, метаданные*

Введение

Подход к построению программных систем на основе архитектуры, называемый *Representational State Transfer* (REST или REST-сервисы), является наиболее часто используемым, особенно для веб-приложений [1]. Архитектурная модель REST является довольно простой в реализации и привлекательна для архитекторов программных систем и разработчиков. Она предлагает небольшой набор базовых принципов, которые позволяют реализовывать высокопроизводительные и масштабируемые программные сервисы. К числу таких базовых принципов могут быть отнесены: унифицированный интерфейс; отсутствие необходимости поддержки состояний; понятная структура запросов (в отношении последнего в англоязычной литературе часто используется словосочетание *self-descriptive messages*).

Единое (унифицированное) представление означает наличие одного и того же интерфейса для всех клиентов. В других системах это может быть совсем не так. Например, в сервисной архитектуре (Service Oriented Architecture — SOA), с которой чаще всего сравнивают REST, для каждого из ресурсов может быть задан свой собственный интерфейс [2]. Именно персонализация вызывает необходимость сохранять информацию о состоянии в SOA-запросах. Технология REST (что особенно важно в связи с ростом нагрузки в сервисах) хорошо масштабируется и характеризуется "прозрачными" принципами оценки производительности. Унифицированное наименование ресурсов также относится к числу базовых принципов REST-модели. Уникальные идентификаторы ресурсов в модели REST (URI) просты для понимания и использования в работе. Перечисленные качества обеспечивают бóльшую популярность подхода REST по сравнению с SOA. Например, все программные интерфейсы для популярных Интернет-сервисов (Facebook, Twitter и т. п.) реализованы на основе REST-архитектуры. Подход на основе REST-архитектуры лежит в основе моделей, которые предлагаются для *Internet of Things* (*Web of Things*) [3]. Вместе с тем в модели REST отсутствует один важный элемент, присущий архитектуре SOA, а именно — метаданные. Необходимость описывать различные ресурсы и поддерживать механизмы персонализации доступа (обращения) привела к тому, что в SOA изначально присутствовал элемент, который описывал сами данные. Описание сервисов — неотъемлемая часть SOA, и *Web Service Definition Language* (WSDL) [4] есть неотъемлемая часть спецификации любого веб-сервиса. Утверждение о том, что различные сервисы могут иметь разные интерфейсы — базовое положение SOA. Соответственно, без наличия описания таких интерфейсов система функционировать не может. Язык WSDL позволяет описывать интерфейсы в терминах входящих и выходящих сообщений. Напротив, вся идея REST-модели построена на простых сообщениях, которые не требуют описания и, как следствие, метаданные в REST-модели отсутствуют. Они не являются частью процесса (модели) как в SOA. Отказ от метаданных при введении REST-подхода представлялся как его преимущество. В каких-то случаях это верное утверждение. Так, объем подготавливаемых данных в REST-модели существенно меньше, чем, например в SOA. Вместе с тем нельзя не отметить, что есть достаточно примеров, когда наличие метаданных просто необходимо,

например, во всех проектах, связанных с автоматизацией программирования (автоматизацией использования API). Такая автоматизация возможна только при наличии формальных описаний программных интерфейсов. Генератор программ может получить исходную информацию об используемых программных интерфейсах только из некоторого формального описания этих интерфейсов. Именно такое формальное описание и представляет собой метаданные. Самым очевидным примером, где использование автоматизации программирования будет необходимо, являются приложения для *Internet of Things* (IoT) или для взаимодействия *Machine to Machine* (M2M) [5]. Большое число устройств и, как следствие, большое число программных интерфейсов затруднит (сделает в итоге невозможным) ручное кодирование. Таким образом, задача представления метаданных для REST-интерфейсов является актуальной. Настоящая работа содержит краткий анализ подходов к представлению метаданных в REST-модели.

Роль метаданных в REST-модели

В первую очередь следует отметить, какая информация может быть представлена в качестве метаданных для модели REST. Список потенциальных кандидатов, на самом деле, следует из описания WSDL (рис. 1). Например, самый очевидный кандидат на представление в наборе метаданных — это точка доступа (в WSDL 1.1 — Port). В случае SOA-модели разработчики могут получить адрес для обращения к веб-сервису из документа WSDL. В случае REST API такой возможности нет. Как правило, описание API для REST-модели представляет собой текстовый (HTML) документ.

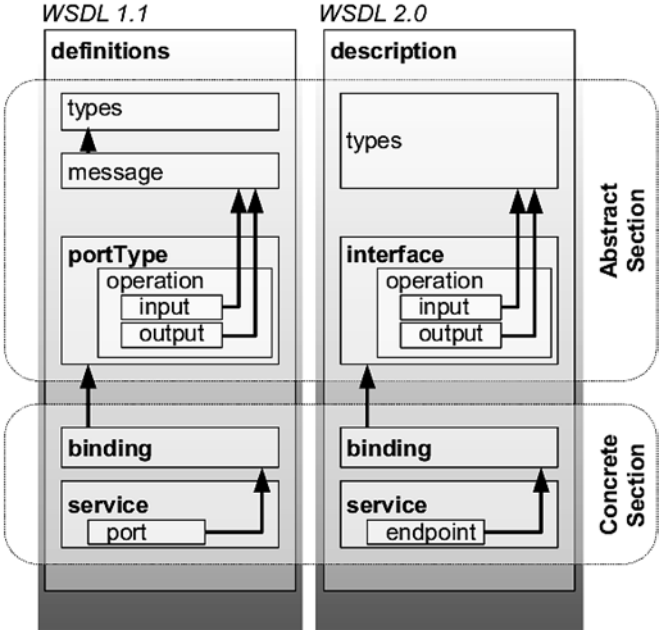

Рис. 1. Структура WSDL-документа [6]

В целом можно выделить предложенные далее элементы для метаданных в REST-модели.

1. Описание точек доступа, URLs для запросов, пути для ресурсов.
2. Методы для доступа к ресурсам. Это HTTP-команды, однако каким-то образом необходимо понимать, например, где использовать HTTP GET, а где — HTTP POST.
3. Параметры в запросах.
4. Заголовки HTTP. Они часто используются в модели REST, например, для определения типов данных.
5. Коды возвратов и сообщения об ошибках.
6. Поддержка версий.
7. Форматы для результатов запросов. По частоте использования JSON-формат, несомненно, превалирует в представлении результатов, однако единственно возможным не является.

Можно отметить, что для некоторых из перечисленных выше элементов существуют уже устоявшиеся решения (своего рода де-факто стандарты), которым негласно следует большинство реализаций. В частности, задание версий часто проводится в URL для запроса. Например, обращение к Twitter API:

https://api.twitter.com/1.1/statuses/mentions_timeline.json,

здесь 1.1 — это и есть номер версии.

Другой общепринятый (повсеместно используемый) элемент, который также присутствует в данном примере, — это расширение. Оно используется в URL и описывает формат результата. В данном случае (расширение *.json*) — это формат JSON, а не XML. Вместе с тем необходимо заметить, что все представленные выше примеры не более чем "псевдометаданные", которые не могут служить полноценной заменой (аналогом) WSDL в SOA.

Поддержка метаданных в подходе REST

Далее остановимся на программных средствах, поддерживающих определение метаданных в REST-моделях.

Превалирующей на настоящее время идеей является разметка кода (аннотации), которая может быть использована для автоматической генерации документации, клиентского кода и тестовых примеров. К такого рода продуктам относятся, например, Swagger [7] и API Blueprint [8]. Программное средство Swagger создает JSON-файлы по аннотированному серверному коду. С его помощью может также создаваться клиентский код на различных языках программирования. Средство API Blueprint в первую очередь ориентировано на создание интерактивной документации. Далее представлен пример разметки кода (описания). Эта разметка (аннотации) и используется для автоматического создания программной документации:

```
FORMAT: 1A
HOST: http://api.gtdtodoapi.com
# GTD TODO API
This is an example API, written as a companion to a blog post at SendGrid.com
## Folder [/folder{id}]
A single Folder object, it represents a single folder.
Required attributes:

- `id` Automatically assigned
- `name`
- `description`
Optional attributes:
- `parent` ID of folder that is the parent. Set to 0 if no parent
- `meta` A catch-all attribute to add custom features
+ Parameters
+ id (required, int)... Unique folder ID in the form of an integer
+ Model (application/hal+json)
+ Body
{
"id": 1,
"name": "Health",
"description": "This represents projects that are related to health"
"parent": 0,
"meta": "NULL"
}
```

Результатом обработки такого кода становится готовая к публикации документация. Она представляет собой набор HTML-файлов (XML с соответствующими стилями). Ниже приведен пример результата их отображения.

```
Function: Retrieve a single Folder
Command: GET
Path: /folder{id}
Parameters Name: id
Description Details: Unique folder ID in
the form of an integer
Type:     int, required
```

По существу, система работает как генератор статических веб-сайтов. Системы, базирующиеся на разметке кода, не ставят своей целью создание некоторого репозитория с описанием API. Их основная задача — помощь конечным разработчикам, которые сами пишут код, а не автоматизация программирования. Другой вопрос, который в связи с этим возникает, это отсутствие стандарта на такую разметку. Если следовать модели (архитектуре) SOA, то синтаксис разметки следовало бы как-то стандартизовать. Это по крайней мере позволило бы переиспользовать код в разных проектах.

Следующий пример — это системы, которые ориентированы на разработку API и поддержку всего жизненного цикла работы с ними (документирование, тестирование, анализ производительности и т. д.). Типичный пример — Anypoint Platform [9]. Эта система включает средства для создания (дизайна) API, тестирования, кодогенерации, шлюз для запуска разработанного REST API, а также средства сбора статистики и мониторинга (рис. 2).

Сложность такого рода портала (порталов) состоит в том, что любая практическая разработка должна не только создавать новые API, но и оперировать с несколькими уже существующими. Применительно к веб-приложениям такой подход (одновременное использование нескольких API) часто называют мэшап. В этом случае подключение к закрытой системе управления новых API — это отдельная задача. Она как-то может быть решена самим производителем программного продукта за счет выпуска набора модулей подключения (адаптеров), например, для по-

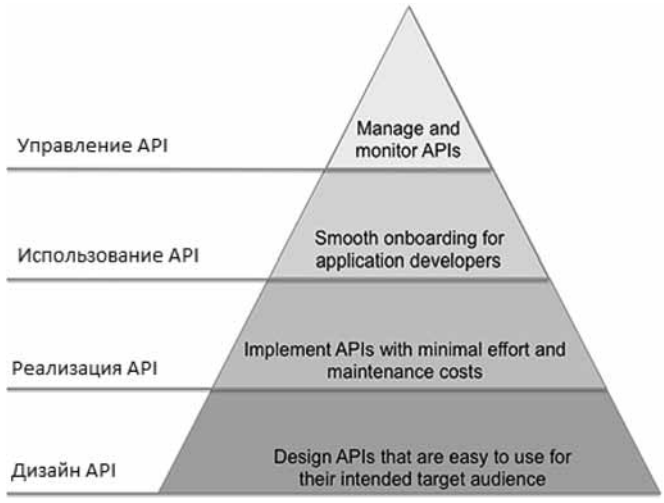

Рис. 2. Полная система поддержки API [10]

пулярных веб-интерфейсов (Facebook, Twitter и т. д.). Однако вопрос подключения остается открытым для множества API в M2M и IoT-приложениях. Сам по себе такой портал представляет собой закрытую систему. Например, существует используемый внутри системы язык разметки (аннотаций), однако он также не является каким-либо стандартом. Аннотированный в данной системе код нельзя использовать в другом приложении. В этом смысле система ничем не отличается от описанного выше средства Blueprint.

Фактически, в таком портале по управлению API метаданные для REST API создаются, однако их использование носит внутренний характер. Все ограничено рамками самого портала. Из других похожих систем можно назвать Intel Mashery [11], которая также представляет собой платформу для управления API.

Следующая группа инструментальных средств — это языки описания. К их числу можно отнести WADL [12], HAL [13], RSDL [14].

HAL (*Hypertext Application Language*) представляет собой текстовый документ (JSON или XML) для описания ссылок и ресурсов. Если для описания API используется HAL, то это описание должно помочь в определении того, по каким ссылкам необходимо обращаться к тем или иным ресурсам. Ниже приведен пример HAL-документа.

```
{
   " _ links": {
      "self": { "href": "/orders/124" },
      "ea:basket": { "href": "/baskets/97213"
},
      "ea:customer": { "href":
"/customers/12369" }
},
      "total": 20.00,
      "currency": "USD",
      "status": "processing"
   }
```

В этом фрагменте описаны ссылки (*self, basket, customer*) и ресурсы (*total, currency, status*).

Web Application Description Language (WADL) предлагает некоторую схему XML для описания веб-приложений. Наиболее близкий аналог подобного подхода — WSDL. Такой XML-документ не зависит от платформы. Рассмотрим пример того, как выглядит описание сервиса на WADL.

```
<resources
base="http://api.search.yahoo.com/NewsSearchService/V1/
">
     <resource path="newsSearch">
        <method name="GET" id="search">
           <request>
              <param name="appid" type="xsd:string"
                 style="query" required="true"/>
              <param name="type" style="query" default="all">
                 <option value="all"/>
                 <option value="any"/>
              </param>
              <param name="results" style="query" type="xsd:int" default="10"/>
           </request>
           <response status="200">
              <representation mediaType="application/xml"
                 element="yn:ResultSet"/>
           </response>
           <response status="400">
              <representation mediaType="application/xml"
                 element="ya:Error"/>
           </response>
        </method>
     </resource>
  </resources>
```

Здесь указан запрос (*request*), параметры и описан отклик (*response*). Язык WADL был предложен как возможный стандарт консорциума W3C. Формально, эта спецификация не изменялась с 2009 г. [15]. Имея спецификацию WADL, можно, естественно, автоматически создавать программные клиенты. Есть готовые решения для такого рода продуктов (Oracle [16]). Решение обратной задачи (создание спецификаций по некоторому программному коду) снова требует использования некоторой разметки. Упомянутое выше решение от Oracle ориентировано на язык программирования Java и использует разметку, соответствующую спецификациям JavaDoc, что иллюстрирует приведенный далее код.

```
// The Java class will be hosted at the URI
// path "/helloworld"
@Path("/helloworld")
public class HelloWorldResource {

   // The Java method will process HTTP GET
   // requests
   @GET
   // The Java method will produce content
   //identified by the MIME Media
   // type "text/plain"
   @Produces("text/plain")
   public String getClichedMessage() {
      // Return some cliched textual content
      return "Hello World";
      }
   }
```

Курсивом выделены элементы, которые могут быть использованы для генерации WADL-файла. Естественно, что это решение только для языка Java, и разметка не совместима, например, с упомянутым выше API Blueprint. Для языка Java эти аннотации описывались соответствующей спецификацией JSR-311 [17]. По факту (на практике) сообществом разработчиков WADL не принят и массового применения не имеет. Доказательством этого утверждения служит факт отсутствия формальных спецификаций для массово используемых REST API (Facebook, Twitter). По мнению автора, WADL мог бы стать самым реальным кандидатом на роль переносимого стандарта для представления метаданных в REST-моделях.

RESTful Service Description Language (RSDL) также предлагает использовать XML для описания веб-приложений. Можно считать, что в этом случае реализуется более простая схема, чем WADL. Для каждого ресурса (URI) в RSDL описывается формат HTTP-запроса (заголовки и параметры), а также соответствующий отклик. Далее приведен фрагмент RSDL-описания.

```
<general rel="*" href="/*">
   <request>
      <headers>
         <header required="true|false">
            <name />
            <description />
            <value />
         </header>
      </headers>
      <url>
         <parameters _ set>
         <parameter context="query|matrix"
type="xs:string"
            required="true|false">
            <name />
            <value />
         </parameter>
      </parameters _ set>
   </url>
   </request>
   <name />
   <description />
</general>
```

Следует отметить, что RSDL также на настоящее время не получил широкого распространения.

Заключение

В настоящей работе проанализированы различные программные средства (продукты), которые применяются или могут применяться для поддержки задания метаданных в приложениях, использующих архитектуру REST. Можно заключить, что на настоящее время не существует некоторого единого стандарта для решения таких задач. Такого стандарта для представления метаданных в модели REST не существует как де-юре (нет никакой единой стандартной спецификации), так и де-факто (нет какого-то лидирующего продукта/подхода, используемого большинством разработчиков). Вместе с тем представляется, что задача описания метаданных для REST-модели является весьма актуальной. Основным движущим мотивом к ее решению является необходимость автоматизации программирования и верификации созданных программ. Наибольшую актуальность этот вопрос приобретет в приложениях, связанных с большим количеством интерфейсов, основанных на REST-архитектуре. Как потенциально важные на перспективу можно рассматривать M2M и IoT-приложения. Как следствие, высока вероятность того, что появятся некоторые де-факто стандартные решения по представлению метаданных для REST-модели, ориентированные на конкретный класс моделей (конкретную прикладную область).

Список литературы

1. **Riva C., Laitkorpi M.** Designing web-based mobile services with REST // Service-Oriented Computing-ICSOC 2007 Workshops. Springer Berlin Heidelberg, 2009. P. 439—450.
2. **Monfort V., Hammoudi S.** Towards Adaptable SOA: Model Driven Development, Context and Aspect //Service-Oriented Computing-ICSOC 2007 Workshops. Springer Berlin Heidelberg, 2009. P. 175—189.
3. **Namiot D., Sneps-Sneppe M.** On IoT Programming //International Journal of Open Information Technologies. 2014. Vol. 2, N. 10. P. 25—28.
4. **Curbera F., Duftler M., Khalaf R. et al.** Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI //IEEE Internet computing. 2002. Vol. 6, N. 2. P. 86—93.
5. **Namiot D., Sneps-Sneppe M.** On M2M Software //International Journal of Open Information Technologies. 2014. Vol. 2, N. 6. P. 29—36.
6. **WSDL** Tutorial. URL: http://www.teqlog.com/wsdl-example-explained-step-by-step.html
7. **Izquierdo J. L. C., Jordi C.** Composing JSON-Based Web APIs. //14th International Conference. ICWE 2014: Toulouse, France. July 1—4, 2014. Proceedings. Springer International Publishing, 2014. P. 390—399.
8. **Apiary**. URL: http://apiary.io
9. **Li W., Svard P.** REST-based SOA application in the cloud: a text correction service case study// In Services (SERVICES-1), 2010 6th World Congress on. IEEE, 2010. P. 84—90.
10. **API** Management. URL: http://blogs.mulesoft.org/raml-apikit-api-hierarchy/
11. **Raivio Y., Luukkainen S., Seppala S.** Towards Open Telco-Business models of API management providers //System Sciences (HICSS), 2011 44th Hawaii International Conference on. IEEE, 2011. P. 1—11.
12. **Hadley M. J.** Web application description language (WADL). Technical Report, Sun Microsystems, Inc. Mountain View, CA, USA, 2006.
13. **Kelly M.** JSON Hypertext Application Language. IETF Draft. URL: https://tools.ietf.org/html/draft-kelly-json-hal-02
14. **Pasternak M.** Restful service description language. U.S. Patent Application 13/656,032.
15. **Спецификация** WADL. URL: http://www.w3.org/Submission/wadl/
16. **Project** Jersey. URL: http://docs.oracle.com/cd/E19776-01/820-4867/book-info/index.html
17. **JSR-311.** URL: https://jcp.org/en/jsr/detail?id=311

D. E. Namiot, Senior Research Fellow, e-mail: dnamiot@gmail.com,
Lomonosov Moscow State University

Metadata in REST Models

This paper discusses the metadata support in software models based on Representational State Transfer (REST) approach. REST is, without a doubt, the most commonly used software architecture model. REST is simpler and often more attractive to developers, comparing with the Service Oriented Architecture. However, this simplification has led to the fact that the REST architecture skips such an element as a classic model of metadata. Consideration of why the metadata may be needed, and which tools can represent them in the REST architecture is the subject of this article. We specify the possible areas for metadata description in REST. The paper discusses the various existing approaches to the provision of metadata in REST, such as Blueprint API, Apiary, Mashery, WADL, HAL and RSDL. The most prospect areas for metadata in REST are Internet of Things and Machine to Machine applications.

Keywords: *web service, SOAP, REST, metadata*

References

1. **Riva C., Laitkorpi M.** Designing web-based mobile services with REST. *Service-Oriented Computing-ICSOC 2007 Workshops.* Springer Berlin Heidelberg, 2009, pp. 439—450.
2. **Monfort V., Hammoudi S.** Towards Adaptable SOA: Model Driven Development, Context and Aspect. *Service-Oriented Computing-ICSOC 2007 Workshops.* Springer Berlin Heidelberg, 2009, pp. 175—189.
3. **Namiot D., Sneps-Sneppe M.** On IoT Programming. *International Journal of Open Information Technologies,* 2014. vol. 2, no. 10, pp. 25—28.
4. **Curbera F., Duftler M., Khalaf R. et al.** Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. *IEEE Internet computing*, 2002, vol. 6, no. 2, pp. 86—93.
5. **Namiot D., Sneps-Sneppe M.** On M2M Software. *International Journal of Open Information Technologies*, 2014. vol. 2, no. 6, pp. 29—36.
6. **WSDL** Tutorial, available at: http://www.teqlog.com/wsdl-example-explained-step-by-step.html
7. **Izquierdo J. L. C., Jordi C.** Composing JSON-Based Web APIs. *14th International Conference. ICWE 2014 Proceedings.* Toulouse, France. July 1—4, 2014. Springer International Publishing, 2014, pp. 390—399.
8. **Apiary**, available at: http://apiary.io
9. **Li W., Svard P.** REST-based SOA application in the cloud: a text correction service case study. *In Services (SERVICES-1), 2010 6th World Congress on*. IEEE, 2010, pp. 84—90.
10. **API** Management, available at: http://blogs.mulesoft.org/raml-apikit-api-hierarchy/
11. **Raivio Y., Luukkainen S., Seppala S.** Towards Open Telco-Business models of API management providers. *System Sciences (HICSS), 2011 44th Hawaii International Conference on*. IEEE, 2011, pp. 1—11.
12. **Hadley M. J.** Web application description language (WADL). Technical Report, Sun Microsystems, Inc. Mountain View, CA, USA 2006.
13. **Kelly M.** JSON Hypertext Application Language. IETF Draft, available at: https://tools.ietf.org/html/draft-kelly-json-hal-02
14. **Pasternak M.** Restful service description language. U.S. Patent Application 13/656,032.
15. **WADL** Specification, available at: http://www.w3.org/Submission/wadl/
16. **Project** Jersey, available at: http://docs.oracle.com/cd/E19776-01/820-4867/book-info/index.html
17. **JSR-311,** available at: https://jcp.org/en/jsr/detail?id=311

УДК 004.4

В. В. Костюк, канд. техн. наук, доц., e-mail: kvv@rguitp.ru, **М. В. Балцану**, магистрант,
Институт инновационных технологий и предпринимательства МГУТУ

Анализ производительности современных технологий взаимодействия приложений с реляционными базами данных

Отражены результаты научно-исследовательской работы по оценке производительности отдельных современных технологий взаимодействия приложений с реляционными базами данных. Были исследованы три технологии: прямое выполнение SQL-запросов, объектно-реляционное проецирование и динамическое формирование и исполнение запросов. Представлены конфигурация тестирующего стенда и условия тестирования, приведены результаты исследования и анализ этих результатов.

Ключевые слова: *технологии программирования, конфигурирование, тестирование, реляционные базы данных, веб-сервисы*

На настоящее время существует большое число приложений (программных компонентов и систем), которые в процессе своего функционирования взаимодействуют с реляционными базами данных. Такое взаимодействие может осуществляться с использованием различных технологий. Каждая из технологий взаимодействия с реляционными базами данных обладает своими показателями производительности.

Современные подходы разработки информационных систем, такие как клиент-серверные конфигурации, сервис-ориентированная архитектура или модульный подход, позволяют использовать различные технологии взаимодействия с реляционными базами данных [1]. В тех модулях информационной системы, где требуется высокая производительность, целесообразно использовать технологии, позволяющие добиться такого уровня.

Целью настоящей статьи является представление результатов исследования, направленных на получение оценки производительности современных технологий взаимодействия приложений с реляционными базами данных в рассмотренных далее условиях.

Следует отметить, что полученные результаты не являются обобщением для их использования в условиях, отличных от тех, которые описаны в статье.

Средства взаимодействия приложений с реляционными базами данных

В этом разделе представлены средства [2], которые используются в качестве примера реализации приложений уровня предприятия с применением конкретных технологий, участвующих в исследовании. В качестве таких средств рассматриваются:

- прямое выполнение SQL-запросов;
- объектно-реляционное проецирование;
- динамическое формирование и исполнение запросов.

Прямое выполнение SQL-запросов

Одна из особенностей языка SQL — независимость от конкретной системы управления базой данных (СУБД), несмотря на наличие различных диалектов языка. Все основные операторы выполняются одинаково в различных СУБД. Существуют лишь особенности специфики языка, которые у каждой СУБД учитываются по-разному. Представителем подхода на основе прямого выполнения SQL-запросов является технология JDBC (*Java DataBase Connectivity*), которая представляет собой интерфейс для организации доступа Java-приложений к базам данных.

Объектно-реляционное проецирование

Одним из широко используемых подходов к организации взаимодействия с реляционными базами данных является объектно-реляционное проецирование (*Object-Relational Mapping*, ORM). Его целью является освобождение разработчика от значительного объема сравнительно низкоуровневого программирования по обеспечению хранения объектов в реляционной базе данных. Предоставляя определенный интерфейс для работы с данными, разработчик не должен заботиться о том, где и в каком виде хранятся данные. Он получает удобные программные механизмы для работы с объектом,

который представляет собой некоторый набор данных. Такие механизмы не только существенно сокращают время разработки программного продукта, но и влекут за собой определенные затраты. Кроме того обстоятельства, что от разработчика требуется хорошее знание определенного средства разработки на основе ORM, при его использовании в крупном приложении существенно ощущается потеря производительности. Представителем таких средств является технология Hibernate — API-библиотека для языка программирования Java, предназначенная для решения задач объектно-реляционного отображения данных.

Средства динамического формирования и исполнения запросов

Для снижения трудоемкости использования прямого выполнения SQL-запросов активно используют различные библиотеки программ. Они выступают как некоторое среднее решение между объектно-реляционным проецированием и прямым выполнением SQL-запросов. На настоящее время существует несколько таких библиотек, каждая их которых призвана разрешать вопросы трудоемкости разработки и сопровождения средств взаимодействия с базами данных, а также повышения их производительности. Разрешение этих вопросов осуществляется за счет создания механизмов динамического формирования SQL-запросов, массового их исполнения, разделения кода приложения и текста запросов.

Одной из таких технологий является технология Apache Cayenne. Несмотря на то что Apache Cayenne является современным средством разработки программ на основе ORM, это средство позволяет одновременно использовать собственные SQL-запросы с объектно-реляционным проецированием. За счет такого подхода у разработчиков появляется возможность заменить те места приложения, где объектно-реляционное проецирование показывает плохую производительность, на собственные оптимизированные динамические SQL-запросы за счет тесной интеграции с технологией Apache Velocity [3]. Более того, Apache Cayenne предоставляет возможность отказаться от конструирования сложных Java-объектов, позволяя получать результаты выполнения запросов в виде карт (Map<String, Object>). Такая возможность широко используется в тех частях исходного кода приложения, где к результатам выполнения запроса к базе данных не применяется какая-либо бизнес-логика, а данные без предварительной обработки передаются в разные части приложения. Технология Apache Cayenne предоставляет также собственные гибкие средства взаимодействия с различными источниками данных, а именно — хранилище запросов, средства массового исполнения запросов и возможность исполнения постраничного вывода информации.

Класс DataContext для большинства приложений является основной точкой доступа к функциям Apache Cayenne. Он предоставляет возможности для исполнения запросов и управления состоянием объектов DataObject.

Условия проведения тестовых испытаний

Исследования, результаты которых представлены далее, проводились с использованием одинаковых приложений, каждое из которых отличалось только технологией взаимодействия с реляционной базой данных, а также путем проведения нагрузочного тестирования каждого из приложений.

В качестве такого приложения был разработан веб-сервис [4], реализующий бизнес-процесс информационной системы уровня предприятия для каждой из упомянутых выше технологий взаимодействия с реляционными базами данных. В качестве предметной области автоматизации была выбрана банковская деятельность и один из ее основных бизнес-процессов — процесс "Прием перевода через платежную систему".

Время обработки запроса пользователя к базе данных можно разделить на время обработки запроса сервером приложений и сервером базы данных. В качестве подлежащего тестовым испытаниям объекта данных (бизнес-объекта, БО) использовалось отношение (свободная таблица — *free table*) "Перевод" базы данных под управлением СУБД DB2 выбранной предметной области (рис. 1).

При разработке архитектуры веб-сервиса использовались следующие технологии и инструментальные средства:

- Java — объектно-ориентированный язык программирования;
- JUnit — библиотека для модульного тестирования программного обеспечения на языке Java;
- DBUnit — библиотека Java, широко используемая в модульном тестировании;
- IntelliJIDEA 11.1 CommunityEdition— интегрированная среда (в качестве среды разработки);
- IBMDataStudio — интегрированное программное решение для управления данными;
- Wsimport — jax-ws-инструмент, позволяющий генерировать различные переносимые артефакты веб-сервиса на основе WSDL-файла;
- SoapUI — инструментарий для тестирования SOA-приложений;
- DbVisualizer — инструмент для разработчиков и администраторов баз данных.

Для реализации генератора вызовов веб-сервиса было разработано клиентское приложение, предоставляющее возможность вызова всех API веб-сервиса. Для каждого из трех упомянутых выше подходов взаимодействия приложениия с реляционными базами данных определены следующие четыре стереотипа API:

- APIitbuTransferMassInsert (вставки записей);
- APIitbuTransferMassUpdate (модификации записей);
- APIitbuTransferFindListByID (поиска записей);
- APIitbuTransferDeleteByListID (удаление записей).

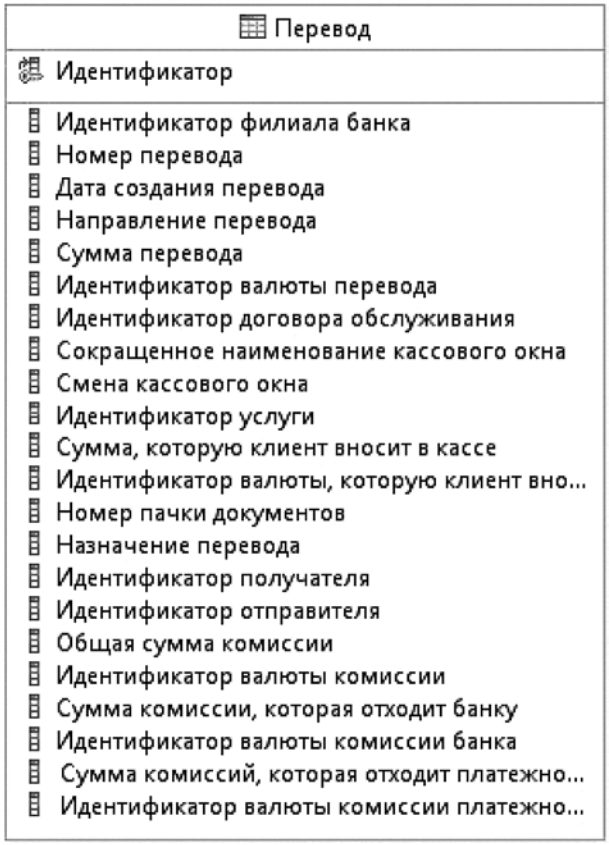

Рис. 1. Схема подлежащего тестовым испытаниям объекта данных

Реализация генератора вызовов построена на основе порождающих шаблонов проектирования фабрик (*Factory*) и строителей (*Builder*). Для порождения входящих наборов данных (готовых данных для вызова API) реализованы классы-фабрики. Для каждого стереотипа API реализованы фабрики, порождающие по шесть наборов данных, содержащих, как минимум, 10 % невалидных элементов данных (1, 10, 100, 1000, 10 000, 50 000 соответственно). Невалидный набор данных — это такой набор данных, после которого исполнение API завершится с ошибкой. В табл. 1 представлен пример реализованных фабрик для стереотипа API APIitbuTransferMassInsert.

Таблица 1

Входящие наборы данных, порождаемые фабриками

Фабрика	Число валидных наборов данных	Число невалидных наборов данных
Insert1Factory	1	1
Insert10Factory	10	1
Insert100Factory	100	10
Insert1000Factory	1000	100
Insert10000Factory	10000	1000
Insert50000Factory	50000	5000

Выполнение нагрузочных тестов проводилось в одинаковых условиях.

Тесты выполнялись на одном и том же стенде, с одним и тем же программным и аппаратным обеспечением. Во время проведения тестовых испытаний на стенде функционировало программное обеспечение, которое необходимо для обеспечения работы веб-сервиса.

Конфигурация тестового стенда, на котором осуществлялось нагрузочное тестирование, представлена на рис. 2.

Аппаратная конфигурация тестового стенда

Процессор	Оперативная память	Жёсткий диск	Сетевой адаптер	Коммутатор ЛВС
Тип: Intel Core i7 Кол-во ядер: 4 Частота: 2400 МГц	Тип: DDR3. Объём: 8192МБ	Тип: HDD. Интерфейс: Serial ATA. Объём: 1ТБ Скорость вращения: 7200 об/мин	Realtek PCIe NIC 10/100/1000 Mb Ethernet	10/100/1000 Mb Ethernet

Программная конфигурация тестового стенда

Операционная система	Java Virtual Machine	Сервер приложений	СУБД	JDBC-драйвер
Windows 7 64-bit	jdk-6u41-windows-x64	Oracle Web Logic Server 12c	IBM DB2 Express-C 10.5 for Windows 64-bit	IBM db2jcc-3.50.152

Рис. 2. Конфигурация тестового стенда

При проведении процедуры нагрузочного тестирования соблюдались следующие требования:

— время выполнения вызова API фиксировалось на клиентской стороне;

— время выполнения вызова API фиксировалось с точностью до 1 мс;

— нагрузочные тесты выполнялись многократно, они представляли собой непрерывную последовательность вызовов API веб-сервиса.

Время выполнения запроса (*request response time*) приложением остается одним из самых главных показателей производительности системы или приложения. Это время измеряется на серверной стороне, как показатель времени, которое требуется серверной части для обработки запроса, а также на клиентской стороне, как показатель полного времени, которое необходимо на сериализацию и (или) десериализацию, пересылку и обработку запроса. Следует заметить, что не каждое приложение для тестирования производительности может измерить оба этих показателя времени.

Фиксирование времени на серверной стороне широко используется для анализа производительности приложения на уровне сервиса и на уровне доступа к данным. Факт фиксации времени на клиентской стороне позволяет анализировать производительность на всех трех уровнях приложения — представления, сервиса и доступа к данным.

Анализ полученных результатов

В табл. 2 представлены результаты эксперимента по проведению нагрузочного тестирования. Приведены оценки производительности упомянутых ранее современных технологий взаимодействия приложений с реляционными базами данных.

Полученные результаты можно интерпретировать следующим образом.

Для APIitbuTransferMassInsert добавление БО "Перевод" как на малых объемах данных, так и на больших, технологии JDBC и Hibernate показали практически одинаковые результаты. Так, среднее время добавления одного БО составило 12,8 мс для JDBC и 13,2 мс для Hibernate, а для 50 000 БО — 85 804 и 85 639,5 мс соответственно. Технология Cay-

Таблица 2

Сводная таблица результатов проведения нагрузочного тестирования

Технология взаимодействия с БД	Число валидных наборов данных					
	1	10	100	1000	10000	50000
	Время исполнения, мс					
APIitbuTransferMassInsert						
JDBC	12,8	29,5	227,6	5767,0	39 031,5	85 804,0
Cayenne	20,1	68,7	452,7	4011,0	44 136,4	223 012,1
Hibernate	13,2	33,3	179,1	1536,8	30 700,3	85 639,5
APIitbuTransferMassUpdate						
JDBC	10,9	20,0	262,7	1915,6	30 592,7	142 287,6
Cayenne	23,6	49,4	273,1	2184,2	20 523,0	113 801,9
Hibernate	17,8	45,9	215,1	1579,4	11 690,6	49 064,7
APIitbuTransferFindListByID						
JDBC	9,7	15,9	55,0	428,9	2028,7	3480,2
Cayenne	14,6	21,7	72,4	442,6	618,7	1366,5
Hibernate	19,7	54,3	241,9	1700,2	31117,9	1 084 526,0
APIitbuTransferDeleteByListID						
JDBC	10,5	10,8	21,9	210,3	1980,1	88 677,2
Cayenne	10,6	11,0	22,1	218,1	2248,3	113 801,9
Hibernate	40,0	210,7	1347,4	12 315,1	60 818,2	1 313 868,0

enne показала худшие результаты на всех объемах. Добавление одного БО составило 20,1 мс, а 50 000 БО — 223 012,1 мс. Следует отметить, что с использованием технологии Hibernate лучшие результаты получены при добавлении БО в объеме от 100 до 10 000.

Для APIitbuTransferMassUpdate изменение БО "Перевод" на объемах в 1 и 10 БО технология JDBC показала наилучший результат 10,9 и 20 мс против 23,6 и 49,4 мс при использовании технологии Cayenne, а также — 17,8 и 45,9 мс при использовании технологии Hibernate. На объемах от 10 000 БО технология Cayenne показала результат лучше, чем технология JDBC.

Для APIitbuTransferFindListByID получение БО "Перевод" на объемах в 1 и 1000 БО технология JDBC показала наилучший результат. Так, на объеме в 1000 БО среднее время исполнения составило 428,9 мс против 442,6 мс при использовании технологии Cayenne и, соответственно, 1700,2 мс при использовании технологии Hibernate. На объемах от 10 000 БО технология Cayenne показала результат лучше, чем JDBC. Среднее время просмотра 50 000 БО составило 1366,5 мс против 3480,2 мс (JDBC) и 1 084 526,0 мс (Hibernate). Технология Hibernate показала худший результат на всех объемах данных.

Для APIitbuTransferDeleteByListID удаление БО "Перевод" на всех объемах технология JDBC показала лучший результат. Среднее время удаления 50 000 БО составило 88 677,2 мс по сравнению с 113 801,9 мс технологии Cayenne и 1 313 868,0 мс технологии Hibernate. Технология Hibernate показала худший результат на всех объемах.

На основании результатов представленных выше экспериментов можно оценить производительность технологий, соответственно, Hibernate, Cayenne и JDBC.

Технология Hibernate для добавления и изменения БО является производительным решением. На объемах от 100 БО эта технология показывает лучшие результаты, а на объемах в 1 и 10 БО результаты уступают только технологии JDBC. Разница во времени исполнения API в случае добавления БО была минимальной — менее 1 мс для 1 БО и 3 мс для 10 БО. В случае изменения БО разница во времени исполнения более существенная — 7 мс для 1 БО и 25 мс для 10 БО. Это связано с тем, что во время исполнения API изменения БО осуществляется проверка существования изменяемого БО. Технология Hiberante показывает очень низкие результаты в случае просмотра БО. Время исполнения больше в 311 раз по сравнению с технологией JDBC и в 793 раза по сравнению с технологией Cayenne. Выполнение API удаления БО, включающей запросы выборки данных, занимает у Hibernate значительно больше времени, чем у других технологий (в 14 раз дольше по сравнению с JDBC, а по сравнению с Cayenne — 11,5 раз).

Использование технологии Hibernate как способа взаимодействия с реляционными базами данных является производительным решением только для реализации функций, осуществляющих минимальное количество выборок данных. Для функций, осуществляющих добавление и изменение записей в базе данных, технология Hibernat является производительным решением.

Технология Cayenne показывает высокие результаты производительности для массового просмотра БО. Так, для просмотра 50 000 БО время исполнения для технологии JDBC больше в 2,5 раза, а для технологии Hibernate — в 793 раза. Наилучше результаты технология Cayenne показывает также для удаления БО на объемах в 1, 10, 100 и 1000 БО. Для удаления и изменения БО технология Cayenne показывает средние результаты на объемах в 10 000 и 50 000 БО. В остальных случаях технология Cayenne является решением с наиболее низким показателем производительности.

Использование технологии Cayenne как способа взаимодействия с реляционными базами данных является производительным решением для реализации функций, осуществляющих максимальное количество выборок данных. Наибольшая производительность этой технологии достигается на больших объемах данных.

Технология JDBC показывает высокие результаты производительности на всех объемах в случае удаления БО. В случае реализации функции, осуществляющей просмотр БО, технология JDBC показывает наилучшие результаты для объема до 1000 БО. Показатели производительности в случае большего объема уступают только технологии Cayenne. Технология JDBC показывает наилучшие результаты производительности в случае добавления и изменения БО в объеме 1 и 10 БО. Наихудшие результаты производительности технологи JDBC получаются только

Таблица 3

Функциональные зоны использования технологий взаимодействия приложений с реляционными базами данных

Функции	Объем данных (число наборов данных)	Технология		
		JDBC	Cayenne	Hibernate
Выборка данных	До 1000	Да	Да	Нет
	Больше 1000	Да	Да	Нет
Добавление данных	До 1000	Да	Нет	Да
	Больше 1000	Да	Нет	Да
Изменение данных	До 1000	Да	Нет	Да
	Больше 1000	Нет	Нет	Да
Удаление данных	До 1000	Да	Да	Нет
	Больше 1000	Да	Да	Нет

для изменения БО на больших объемах (10 000 и 50 000 БО).

Таким образом, использование технологии JDBC как способа взаимодействия с реляционными базами данных является производительным решением для реализации функций, осуществляющих удаление БО для любых объемов данных, а также для выполнения функций, осуществляющих добавление, изменение и просмотр БО в рамках небольшого объема данных.

В табл. 3 на основании приведенных выше оценок представлены функциональные зоны наиболее производительного использования технологий взаимодействия приложений с реляционными базами данных (помечены как Да).

Список литературы

1. **Хорстманн К. С., Корнелл Г.** Java 2. Библиотека профессионала, том 1. Основы. 8-е издание. Пер. с англ. М.: Вильямс, 2009. 816 с.
2. **Хорстманн К. С., Корнелл Г.** Java 2. Библиотека профессионала, том 2. Тонкости программирования, 8-е издание. Пер. с англ. М.: Вильямс, 2009. 992 с.
3. **Lalou J.** Apache Maven Dependency Management. Packt Publishing, 2013. 158 p.
4. **Машнин Т. С**. Web-сервисы Java, Пер. с англ. СПб.: БХВ-Петербург, 2012. 560 с.

V. V. Kostyuk, Professor, e-mail: kvv@rguitp.ru, **M. V. Baltsanu**, Student,
Institute of innovative technologies and enterprise of MSUTM, Moscow

Analysis of the Performance of a Single Modern Technologies to Interact of Applications with Relational Databases

This article reflects the results of research work to assess the performance of a single modern technology for applications to interact with relational databases. Were investigated three approaches: direct execution of SQL queries, object-relational projection and dynamic formation and execution of requests. Presented the configuration of the test stand and the test conditions. As conditions covers such topics as the subject area of business process (banking activities); the data object to be tested; used technologies and tools. For each of the three above approaches for applications to interact with relational databases identified the following four stereotype API: inserting records; modify records; locating records; destruction of records. For each stereotype API implemented mechanisms, generating six datasets containing at least 10% of the obviously erroneous data elements (1, 10, 100, 1000, 10 000, 50 000, respectively). These sets are created on the basis of generation of calls of web-service, worked out Java instrumental with the use of such facilities, as JUnit, DBUnit, IntelliJIDEA 11.1 CommunityEdition, IBMDataStudio, Wsimport (jax-ws) and others.

During the procedure of stress testing observed the requirements for time to repeatedly perform tests. In tabular form the results of the research are presented. Based on the interpretation and analysis of these results, estimates of performance for each of the above stereotypes API and presents the functional areas most productive use of technology for applications to interact with relational databases.

Keywords: *technology, programming, integration, testing, relational databases, web services*

References

1. **Horstmann K. C., Kornell G.** *Java 2. Biblioteka professionala, vol. 1. Osnovy* (Java 2. Library of professional, vol. 1. Bases). Moscow: Williams, 2009. 816 p. (in Russian).
2. **Horstmann K. C., Kornell G.** *Java 2. Biblioteka professionala, vol. 2. Tonkosti programmirovaniya* (Java 2. Library of professional, vol. 2. Subtleties of programming). Moscow: Williams, 2009. 992 p. (in Russian).
3. **Lalou J.** *Apache Maven Dependency Management*. Packt Publishing, 2013. 158 p.
4. **Mashnin T. C.** *Web-servisy Java* (Web-services of Java). Sankt-Petersburg: BHV- Petersburg, 2012. 560 p. (in Russian).

УДК 629.73.02; 629.73.05/.06; 535.643

И. О. Жаринов, д-р техн. наук, доц., зав. каф., Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики (НИУ ИТМО), руководитель учебно-научного центра, Санкт-Петербургское ОКБ "Электроавтоматика" им. П. А. Ефимова, e-mail: igor_rabota@pisem.net
О. О. Жаринов, канд. техн. наук, доц., Санкт-Петербургский государственный университет аэрокосмического приборостроения (ГУАП)

Исследование математической модели цветопередачи жидкокристаллических панелей

Исследована математическая модель цветопередачи жидкокристаллической панели. Приведены выражения для расчета смещения координат цветности изображения на панелях одного и того же, а также различных производителей. Представлены формулы для расчета коэффициентов профиля панели по известным значениям координат вершин треугольника цветового охвата и точки белого цвета. Проанализированы результаты моделирования, подтверждающие возможность описания двумерного распределения координат цветности нормальным распределением, стохастический характер распределения которых обусловлен технологическим разбросом изготовления жидкокристаллических панелей.

Ключевые слова: *координаты цвета, координаты цветности, системы индикации, модель цветопередачи, жидкокристаллическая панель*

Введение

При разработке программного обеспечения бортовых средств индикации класса МФЦИ (многофункциональные цветные индикаторы) специалисты решают задачу выбора цветовой палитры изображения, индицируемого на экране жидкокристаллической (ЖК) панели [1].

Выбор цветовой палитры сопряжен с выполнением требований нормативно-технической документации, действующей в авиационной промышленности, а также с необходимостью представления наблюдателю изображения с наилучшими визуальными характеристиками восприятия. Оценка качества бортового средства индикации выполняется в процессе светотехнических испытаний с использованием инструментальных средств измерения. В результате таких исследований формируется искомая цветовая палитра — массив записей, заданных кодом *RGB* (*R* — Red, *G* — Green, *B* — Blue), для каждого задействованного цвета [2].

Опыт практической разработки МФЦИ показывает, что сформированная однократно цветовая палитра [3], подтвержденная результатами и протоколом светотехнических испытаний одного образца МФЦИ, заимствуется в программное обеспечение (ПО) последующих разработок изделий одного класса. Такой инженерный подход к разработке ПО, аргументированный принципом использования максимально возможных унифицированных и стандартизованных успешных проектных решений, имеет ряд существенных недостатков. При таком подходе, например, не учитываются:

- стохастические свойства координат цветности, обусловленные технологическим разбросом колориметрических характеристик цветопередачи ЖК панелей одного производителя;
- смещение координат цветности, обусловленное разницей в коэффициентах профиля ЖК панелей различных производителей, которые применяют в конструктивных модификациях МФЦИ, изготовленных по одной групповой документации.

В связи с изложенным выше актуальными являются задача исследования математической модели цветопередачи ЖК панели одного производителя и задача определения поправочных коэффициентов, корректирующих компоненты кода *RGB* под свойства цветопередачи ЖК панелей различных производителей [4].

1. Смещение координат цвета и координат цветности на ЖК панелях различных производителей

Для каждой пары МФЦИ, в составе программного обеспечения которых используется единая цветовая палитра, заданная компонентами кода *RGB*, в соответствии с уравнением обратного преобразования Грассмана [5] справедливо

$$\begin{bmatrix} X_{r_2} & X_{g_2} & X_{b_2} \\ Y_{r_2} & Y_{g_2} & Y_{b_2} \\ Z_{r_2} & Z_{g_2} & Z_{b_2} \end{bmatrix}^{-1} \begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \end{bmatrix} = \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} X_{r_1} & X_{g_1} & X_{b_1} \\ Y_{r_1} & Y_{g_1} & Y_{b_1} \\ Z_{r_1} & Z_{g_1} & Z_{b_1} \end{bmatrix}^{-1} \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \end{bmatrix} \Rightarrow \begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \end{bmatrix} = \begin{bmatrix} X_{r_2} & X_{g_2} & X_{b_2} \\ Y_{r_2} & Y_{g_2} & Y_{b_2} \\ Z_{r_2} & Z_{g_2} & Z_{b_2} \end{bmatrix} \begin{bmatrix} X_{r_1} & X_{g_1} & X_{b_1} \\ Y_{r_1} & Y_{g_1} & Y_{b_1} \\ Z_{r_1} & Z_{g_1} & Z_{b_1} \end{bmatrix}^{-1} \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \end{bmatrix} = \begin{bmatrix} \breve{X}_r & \breve{X}_g & \breve{X}_b \\ \breve{Y}_r & \breve{Y}_g & \breve{Y}_b \\ \breve{Z}_r & \breve{Z}_g & \breve{Z}_b \end{bmatrix} \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \end{bmatrix} =$$

$$= \begin{bmatrix} \dfrac{x_2 Y_{xy_2}}{y_2} \\ Y_{xy_2} \\ (1 - x_2 - y_2)\dfrac{Y_{xy_2}}{y_2} \end{bmatrix} = \begin{bmatrix} \breve{X}_r & \breve{X}_g & \breve{X}_b \\ \breve{Y}_r & \breve{Y}_g & \breve{Y}_b \\ \breve{Z}_r & \breve{Z}_g & \breve{Z}_b \end{bmatrix} \begin{bmatrix} \dfrac{x_1 Y_{xy_1}}{y_1} \\ Y_{xy_1} \\ (1 - x_1 - y_1)\dfrac{Y_{xy_1}}{y_1} \end{bmatrix} \Rightarrow$$

$$\Rightarrow \begin{cases} \dfrac{x_2 Y_{xy_2}}{y_2} = \breve{X}_r \dfrac{x_1 Y_{xy_1}}{y_1} + \breve{X}_g Y_{xy_1} + \breve{X}_b (1 - x_1 - y_1)\dfrac{Y_{xy_1}}{y_1} \\ Y_{xy_2} = \breve{Y}_r \dfrac{x_1 Y_{xy_1}}{y_1} + \breve{Y}_g Y_{xy_1} + \breve{Y}_b (1 - x_1 - y_1)\dfrac{Y_{xy_1}}{y_1} \\ (1 - x_2 - y_2)\dfrac{Y_{xy_2}}{y_2} = \breve{Z}_r \dfrac{x_1 Y_{xy_1}}{y_1} + \breve{Z}_g Y_{xy_1} + \breve{Z}_b (1 - x_1 - y_1)\dfrac{Y_{xy_1}}{y_1} \end{cases}, \quad (1)$$

где X_1, Y_1, Z_1 — координаты цвета изображения на экране первого МФЦИ; X_2, Y_2, Z_2 — координаты цвета изображения на экране второго МФЦИ; x_1, y_1, Y_{xy_1} — координаты цветности и компонент относительной яркости изображения на экране первого МФЦИ; x_2, y_2, Y_{xy_2} — координаты цветности и компонент относительной яркости изображения на экране второго МФЦИ; X_{r_i}, X_{g_i}, X_{b_i}, Y_{r_i}, Y_{g_i}, Y_{b_i}, Z_{r_i}, Z_{g_i}, Z_{b_i} — коэффициенты матриц профиля ЖК панелей, установленных в МФЦИ (i = 1, 2); R, G, B — десятичный код цвета.

Система (1) связывает координаты цветности и компоненты относительной яркости изображения для единой цветовой палитры. Отличия в значениях (x, y)-координат цветности, обусловленные разницей в коэффициентах профиля ЖК панелей и вычисляемые в результате решения системы (1), следующие:

$$x_2 = \frac{\breve{X}_r \dfrac{x_1}{y_1} + \breve{X}_g + \breve{X}_b (1 - x_1 - y_1)\dfrac{1}{y_1}}{\left(\breve{X}_r + \breve{Y}_r + \breve{Z}_r\right)\dfrac{x_1}{y_1} + \left(\breve{X}_g + \breve{Y}_g + \breve{Z}_g\right) + \left(\breve{X}_b + \breve{Y}_b + \breve{Z}_b\right)(1 - x_1 - y_1)\dfrac{1}{y_1}}, \quad (2)$$

$$y_2 = \frac{\breve{Y}_r \dfrac{x_1}{y_1} + \breve{Y}_g + \breve{Y}_b (1 - x_1 - y_1)\dfrac{1}{y_1}}{\left(\breve{X}_r + \breve{Y}_r + \breve{Z}_r\right)\dfrac{x_1}{y_1} + \left(\breve{X}_g + \breve{Y}_g + \breve{Z}_g\right) + \left(\breve{X}_b + \breve{Y}_b + \breve{Z}_b\right)(1 - x_1 - y_1)\dfrac{1}{y_1}}. \quad (3)$$

Выражения (2) и (3) определяют правило пересчета координат цветности на XY-плоскости для изображения, заданного одним и тем же кодом RGB, но воспроизводимого на различных моделях индикаторов МФЦИ, в которых установлены ЖК панели различных производителей. Таким образом, для сохранения координат цвета и координат цветности изображения при заимствовании цветовой палитры ПО одного МФЦИ в проект разработки ПО другого МФЦИ необходимо вводить коэффициенты пересчета, определяемые выражением (с округлением каждой из компонент $R_2G_2B_2$ до ближайшего большего целого числа):

$$\begin{bmatrix} X_{r_2} & X_{g_2} & X_{b_2} \\ Y_{r_2} & Y_{g_2} & Y_{b_2} \\ Z_{r_2} & Z_{g_2} & Z_{b_2} \end{bmatrix} \begin{bmatrix} R_2 \\ G_2 \\ B_2 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X_{r_1} & X_{g_1} & X_{b_1} \\ Y_{r_1} & Y_{g_1} & Y_{b_1} \\ Z_{r_1} & Z_{g_1} & Z_{b_1} \end{bmatrix} \begin{bmatrix} R_1 \\ G_1 \\ B_1 \end{bmatrix} \Rightarrow \begin{bmatrix} R_2 \\ G_2 \\ B_2 \end{bmatrix} = \begin{bmatrix} X_{r_2} & X_{g_2} & X_{b_2} \\ Y_{r_2} & Y_{g_2} & Y_{b_2} \\ Z_{r_2} & Z_{g_2} & Z_{b_2} \end{bmatrix}^{-1} \begin{bmatrix} X_{r_1} & X_{g_1} & X_{b_1} \\ Y_{r_1} & Y_{g_1} & Y_{b_1} \\ Z_{r_1} & Z_{g_1} & Z_{b_1} \end{bmatrix} \begin{bmatrix} R_1 \\ G_1 \\ B_1 \end{bmatrix}. \quad (4)$$

Ближайшее большее целое число кода $R_2G_2B_2$ выбирается исходя из принципа соответствия большей яркости изображения более насыщенному цвету, т. е. цвету, заданному большим значением кода RGB.

2. Расчет коэффициентов профиля ЖК панели

Преобразование Грассмана и расчетные соотношения (1)-(4) на его основе предполагают, что коэффициенты X_r, X_g, X_b, Y_r, Y_g, Y_b, Z_r, Z_g, Z_b профиля ЖК панели считаются априори известными. Однако на практике иностранные разработчики экранов приводят в документации, как правило, только координаты вершин треугольника цветового охвата (x_R, y_R), (x_G, y_G), (x_B, y_B) и координаты (x_W, y_W) точки белого цвета. Сами значения коэффициентов профиля производителями в документации не приводятся. Для оценки свойств цветопередачи ЖК панели необходимо проводить

расчет коэффициентов X_r, X_g, X_b, Y_r, Y_g, Y_b, Z_r, Z_g, Z_b
ее профиля.

Расчет коэффициентов профиля предполагает, что (x_R, y_R)-координата вершины треугольника цветового охвата в красном цвете получена на основе прямого преобразования Грассмана [5] по десятичному коду RGB = (255,0,0); (x_G, y_G) — вершина в зеленом цвете для кода RGB = (0,255,0); (x_B, y_B) — вершина в синем цвете для кода RGB = (0,0,255); (x_W, y_W) — точка в белом цвете для кода RGB = (255,255,255). Соответствующие правила вычислений могут быть представлены в виде системы уравнений

$$\begin{cases} x_R\left(\widehat{X}_r+\widehat{Y}_r+\widehat{Z}_r\right)=\widehat{X}_r=\dfrac{x_R}{y_R}\widehat{Y}_r \\ y_R\left(\widehat{X}_r+\widehat{Y}_r+\widehat{Z}_r\right)=\widehat{Y}_r=\dfrac{y_R}{x_R}\widehat{X}_r \\ x_G\left(\widehat{X}_g+\widehat{Y}_g+\widehat{Z}_g\right)=\widehat{X}_g=\dfrac{x_G}{y_G}\widehat{Y}_g \\ y_G\left(\widehat{X}_g+\widehat{Y}_g+Z_g\right)=\widehat{Y}_g=\dfrac{y_G}{x_G}\widehat{X}_g \\ x_B\left(\widehat{X}_b+\widehat{Y}_b+\widehat{Z}_b\right)=\widehat{X}_b=\dfrac{x_B}{y_B}\widehat{Y}_b \\ y_B\left(\widehat{X}_b+\widehat{Y}_b+\widehat{Z}_b\right)=\widehat{Y}_b=\dfrac{y_B}{x_B}\widehat{X}_b \\ x_W\left(\widehat{X}_r+\widehat{X}_g+\widehat{X}_b+\widehat{Y}_r+\widehat{Y}_g+\widehat{Y}_b+\widehat{Z}_r+\widehat{Z}_g+\widehat{Z}_b\right)=\widehat{X}_r+\widehat{X}_g+\widehat{X}_b \\ y_W\left(\widehat{X}_r+\widehat{X}_g+\widehat{X}_b+\widehat{Y}_r+\widehat{Y}_g+\widehat{Y}_b+\widehat{Z}_r+\widehat{Z}_g+\widehat{Z}_b\right)=\widehat{Y}_r+\widehat{Y}_g+\widehat{Y}_b \end{cases}. \quad (5)$$

Система (5) состоит из восьми уравнений и имеет девять неизвестных: $\widehat{X}_r$, $\widehat{Y}_r$, $\widehat{X}_g$, $\widehat{Y}_g$, $\widehat{X}_b$, $\widehat{Y}_b$, $\widehat{Z}_r$, $\widehat{Z}_g$, $\widehat{Z}_b$ — искомые коэффициенты профиля ЖК панели. В существующем виде система (5) не имеет однозначного решения.

Для решения уравнений системы (5) необходимо ввести девятое уравнение, связывающее коэффициенты профиля. Недостающее уравнение имеет следующий вид:

$$\widehat{Y}_r+\widehat{Y}_g+\widehat{Y}_b=1. \quad (6)$$

Связь (6) Y-коэффициентов обусловлена принятым [6] в колориметрии балансом белого цвета, в котором Y-составляющая относительной яркости в белом цвете при коде RGB = (255,255,255) обладает максимальным значением, принимаемым за 100 %.

Выражения (5) и (6) в совокупности образуют следующую систему уравнений:

$$\begin{cases} \widehat{X}_r=\dfrac{x_R}{y_R}\widehat{Y}_r \\ \widehat{Y}_r=\dfrac{y_R}{x_R}\widehat{X}_r \\ \widehat{X}_g=\dfrac{x_G}{y_G}\widehat{Y}_g \\ \widehat{Y}_g=\dfrac{y_G}{x_G}\widehat{X}_g \\ \widehat{X}_b=\dfrac{x_B}{y_B}\widehat{Y}_b \\ \widehat{Y}_b=\dfrac{y_B}{x_B}\widehat{X}_b \\ \widehat{X}_r+\widehat{X}_g+\widehat{X}_b=\dfrac{x_W}{y_W} \\ \widehat{Y}_r+\widehat{Y}_g+\widehat{Y}_b=1 \\ \widehat{Z}_r+\widehat{Z}_g+\widehat{Z}_b=\dfrac{1-x_W-y_W}{y_W} \end{cases} \Rightarrow \begin{cases} \widehat{X}_r=\dfrac{x_R}{y_R}\widehat{Y}_r \\ \widehat{Y}_r=\dfrac{y_R}{x_R}\widehat{X}_r \\ \widehat{X}_g=\dfrac{x_G}{y_G}\widehat{Y}_g \\ \widehat{Y}_g=\dfrac{y_G}{x_G}\widehat{X}_g \\ \widehat{X}_b=\dfrac{x_B}{y_B}\widehat{Y}_b \\ \widehat{Y}_b=\dfrac{y_B}{x_B}\widehat{X}_b \\ \widehat{Z}_r=\widehat{Y}_r\left(\dfrac{1-x_R-y_R}{y_R}\right) \\ \widehat{Z}_g=\widehat{Y}_g\left(\dfrac{1-x_G-y_G}{y_G}\right) \\ \widehat{Z}_b=\widehat{Y}_b\left(\dfrac{1-x_B-y_B}{y_B}\right) \end{cases}, \quad (7)$$

при условии

$$\begin{cases} \dfrac{x_R}{y_R}\hat{Y}_r + \dfrac{x_G}{y_G}\hat{Y}_g + \dfrac{x_B}{y_B}\hat{Y}_b = \dfrac{x_W}{y_W} \\ \hat{Y}_r + \hat{Y}_g + \hat{Y}_b = 1 \\ \hat{Y}_r\left(\dfrac{1-x_R-y_R}{y_R}\right) + \hat{Y}_g\left(\dfrac{1-x_G-y_G}{y_G}\right) + \hat{Y}_b\left(\dfrac{1-x_B-y_B}{y_B}\right) = \dfrac{1-x_W-y_W}{y_W} \end{cases}. \tag{8}$$

Совместное решение систем уравнений (7), (8) относительно неизвестных коэффициентов $\hat{X}_r$, $\hat{Y}_r$, $\hat{X}_g$, $\hat{Y}_g$, $\hat{X}_b$, $\hat{Y}_b$, $\hat{Z}_r$, $\hat{Z}_g$, $\hat{Z}_b$ имеет следующий вид:

$$\hat{X}_r = \frac{x_R}{y_R}\left(1 - \frac{\dfrac{x_W}{y_W} - \dfrac{x_R}{y_R}}{\dfrac{x_g}{y_g} - \dfrac{x_R}{y_R}} - \left(\frac{x_R}{y_R} - \frac{x_B}{y_B} + 1\right)\frac{\dfrac{1-x_W}{y_W} + \dfrac{x_R-1}{y_R} + \dfrac{x_W y_R - y_W x_R}{x_G y_R - y_G x_R}\dfrac{y_G}{y_W}\left(\dfrac{1-x_R}{y_R} + \dfrac{x_G-1}{y_G}\right)}{\dfrac{x_R y_B - y_R x_B}{x_G y_R - y_G x_R}\dfrac{y_G}{y_B}\left(\dfrac{1-x_G}{y_G} + \dfrac{x_R-1}{y_R}\right) + \dfrac{1-x_B}{y_B} + \dfrac{x_R-1}{y_R}}\right), \tag{9}$$

$$\hat{X}_g = \frac{x_G}{y_G}\left(\frac{\dfrac{x_W}{y_W} - \dfrac{x_R}{y_R}}{\dfrac{x_g}{y_g} - \dfrac{x_R}{y_R}} + \left(\frac{x_R}{y_R} - \frac{x_B}{y_B}\right)\frac{\dfrac{1-x_W}{y_W} + \dfrac{x_R-1}{y_R} + \dfrac{x_W y_R - y_W x_R}{x_G y_R - y_G x_R}\dfrac{y_G}{y_W}\left(\dfrac{1-x_R}{y_R} + \dfrac{x_G-1}{y_G}\right)}{\dfrac{x_R y_B - y_R x_B}{x_G y_R - y_G x_R}\dfrac{y_G}{y_B}\left(\dfrac{1-x_G}{y_G} + \dfrac{x_R-1}{y_R}\right) + \dfrac{1-x_B}{y_B} + \dfrac{x_R-1}{y_R}}\right), \tag{10}$$

$$\hat{X}_b = \frac{x_B}{y_B}\frac{\dfrac{1-x_W}{y_W} + \dfrac{x_R-1}{y_R} + \dfrac{x_W y_R - y_W x_R}{x_G y_R - y_G x_R}\dfrac{y_G}{y_W}\left(\dfrac{1-x_R}{y_R} + \dfrac{x_G-1}{y_G}\right)}{\dfrac{x_R y_B - y_R x_B}{x_G y_R - y_G x_R}\dfrac{y_G}{y_B}\left(\dfrac{1-x_G}{y_G} + \dfrac{x_R-1}{y_R}\right) + \dfrac{1-x_B}{y_B} + \dfrac{x_R-1}{y_R}}, \tag{11}$$

$$\hat{Y}_r = 1 - \frac{\dfrac{x_W}{y_W} - \dfrac{x_R}{y_R}}{\dfrac{x_g}{y_g} - \dfrac{x_R}{y_R}} - \left(\frac{x_R}{y_R} - \frac{x_B}{y_B} + 1\right)\frac{\dfrac{1-x_W}{y_W} + \dfrac{x_R-1}{y_R} + \dfrac{x_W y_R - y_W x_R}{x_G y_R - y_G x_R}\dfrac{y_G}{y_W}\left(\dfrac{1-x_R}{y_R} + \dfrac{x_G-1}{y_G}\right)}{\dfrac{x_R y_B - y_R x_B}{x_G y_R - y_G x_R}\dfrac{y_G}{y_B}\left(\dfrac{1-x_G}{y_G} + \dfrac{x_R-1}{y_R}\right) + \dfrac{1-x_B}{y_B} + \dfrac{x_R-1}{y_R}}, \tag{12}$$

$$\hat{Y}_g = \frac{\dfrac{x_W}{y_W} - \dfrac{x_R}{y_R}}{\dfrac{x_g}{y_g} - \dfrac{x_R}{y_R}} + \left(\frac{x_R}{y_R} - \frac{x_B}{y_B}\right)\frac{\dfrac{1-x_W}{y_W} + \dfrac{x_R-1}{y_R} + \dfrac{x_W y_R - y_W x_R}{x_G y_R - y_G x_R}\dfrac{y_G}{y_W}\left(\dfrac{1-x_R}{y_R} + \dfrac{x_G-1}{y_G}\right)}{\dfrac{x_R y_B - y_R x_B}{x_G y_R - y_G x_R}\dfrac{y_G}{y_B}\left(\dfrac{1-x_G}{y_G} + \dfrac{x_R-1}{y_R}\right) + \dfrac{1-x_B}{y_B} + \dfrac{x_R-1}{y_R}}, \tag{13}$$

$$\hat{Y}_b = \frac{\dfrac{1-x_W}{y_W} + \dfrac{x_R-1}{y_R} + \dfrac{x_W y_R - y_W x_R}{x_G y_R - y_G x_R}\dfrac{y_G}{y_W}\left(\dfrac{1-x_R}{y_R} + \dfrac{x_G-1}{y_G}\right)}{\dfrac{x_R y_B - y_R x_B}{x_G y_R - y_G x_R}\dfrac{y_G}{y_B}\left(\dfrac{1-x_G}{y_G} + \dfrac{x_R-1}{y_R}\right) + \dfrac{1-x_B}{y_B} + \dfrac{x_R-1}{y_R}}, \tag{14}$$

$$\hat{Z}_r = \left(\frac{1-x_R-y_R}{y_R}\right)\left(1 - \frac{\dfrac{x_W}{y_W} - \dfrac{x_R}{y_R}}{\dfrac{x_g}{y_g} - \dfrac{x_R}{y_R}} - \left(\frac{x_R}{y_R} - \frac{x_B}{y_B} + 1\right)\frac{\dfrac{1-x_W}{y_W} + \dfrac{x_R-1}{y_R} + \dfrac{x_W y_R - y_W x_R}{x_G y_R - y_G x_R}\dfrac{y_G}{y_W}\left(\dfrac{1-x_R}{y_R} + \dfrac{x_G-1}{y_G}\right)}{\dfrac{x_R y_B - y_R x_B}{x_G y_R - y_G x_R}\dfrac{y_G}{y_B}\left(\dfrac{1-x_G}{y_G} + \dfrac{x_R-1}{y_R}\right) + \dfrac{1-x_B}{y_B} + \dfrac{x_R-1}{y_R}}\right), \tag{15}$$

$$\widehat{Z}_g = \left(\frac{1-x_G-y_G}{y_G}\right)\left(\frac{\frac{x_W}{y_W}-\frac{x_R}{y_R}}{\frac{x_g}{y_g}-\frac{x_R}{y_R}}+\left(\frac{x_R}{y_R}-\frac{x_B}{y_B}\right)\frac{\frac{1-x_W}{y_W}+\frac{x_R-1}{y_R}+\frac{x_Wy_R-y_Wx_R}{x_Gy_R-y_Gx_R}\frac{y_G}{y_W}\left(\frac{1-x_R}{y_R}+\frac{x_G-1}{y_G}\right)}{\frac{x_Ry_B-y_Rx_B}{x_Gy_R-y_Gx_R}\frac{y_G}{y_B}\left(\frac{1-x_G}{y_G}+\frac{x_R-1}{y_R}\right)+\frac{1-x_B}{y_B}+\frac{x_R-1}{y_R}}\right), \quad (16)$$

$$\widehat{Z}_b = \left(\frac{1-x_B-y_B}{y_B}\right)\frac{\frac{1-x_W}{y_W}+\frac{x_R-1}{y_R}+\frac{x_Wy_R-y_Wx_R}{x_Gy_R-y_Gx_R}\frac{y_G}{y_W}\left(\frac{1-x_R}{y_R}+\frac{x_G-1}{y_G}\right)}{\frac{x_Ry_B-y_Rx_B}{x_Gy_R-y_Gx_R}\frac{y_G}{y_B}\left(\frac{1-x_G}{y_G}+\frac{x_R-1}{y_R}\right)+\frac{1-x_B}{y_B}+\frac{x_R-1}{y_R}}. \quad (17)$$

Выражения (9)—(17) определяют правила для вычисления коэффициентов $\widehat{X}_r$, $\widehat{Y}_r$, $\widehat{X}_g$, $\widehat{Y}_g$, $\widehat{X}_b$, $\widehat{Y}_b$, $\widehat{Z}_r$, $\widehat{Z}_g$, $\widehat{Z}_b$ профиля ЖК панели на основе известных значений (x_R, y_R), (x_G, y_G), (x_B, y_B), (x_W, y_W). Такие значения приводятся производителями ЖК панелей в технической документации.

3. Разброс координат цвета и координат цветности на ЖК панелях одного производителя

В соответствии с технической документацией на ЖК панели координаты цветности (x_R, y_R), (x_G, y_G), (x_B, y_B) вершин треугольника цветового охвата задаются в виде интервальных значений, имеющих минимальное, максимальное и типовое (среднее) значение. Примеры задания значений координат вершин треугольника цветового охвата для различных образцов ЖК панелей, определенных изготовителями в документации, приведены в таблице.

Технологический разброс параметров профиля ЖК панели одного производителя учитывается в модели:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X_r+\Delta\xi_{X_r} & X_g+\Delta\xi_{X_g} & X_b+\Delta\xi_{X_b} \\ Y_r+\Delta\xi_{Y_r} & Y_g+\Delta\xi_{Y_g} & Y_b+\Delta\xi_{Y_b} \\ Z_r+\Delta\xi_{Z_r} & Z_g+\Delta\xi_{Z_g} & Z_b+\Delta\xi_{Z_b} \end{bmatrix}\begin{bmatrix} R \\ G \\ B \end{bmatrix} \Rightarrow$$

$$\Rightarrow \begin{cases} X = R\left(X_r+\Delta\xi_{X_r}\right)+G\left(X_g+\Delta\xi_{X_g}\right)+B\left(X_b+\Delta\xi_{X_b}\right) \\ Y = R\left(Y_r+\Delta\xi_{Y_r}\right)+G\left(Y_g+\Delta\xi_{Y_g}\right)+B\left(Y_b+\Delta\xi_{Y_b}\right) \\ Z = R\left(Z_r+\Delta\xi_{Z_r}\right)+G\left(Z_g+\Delta\xi_{Z_g}\right)+B\left(Z_b+\Delta\xi_{Z_b}\right) \end{cases}, \quad (18)$$

где

$$\xi_{X_r} \in \left[-\frac{X_r}{2};+\frac{X_r}{2}\right],\ \xi_{X_g} \in \left[-\frac{X_g}{2};+\frac{X_g}{2}\right],\ \xi_{X_b} \in \left[-\frac{X_b}{2};+\frac{X_b}{2}\right],$$

$$\xi_{Y_r} \in \left[-\frac{Y_r}{2};+\frac{Y_r}{2}\right],\ \xi_{Y_g} \in \left[-\frac{Y_g}{2};+\frac{Y_g}{2}\right],\ \xi_{Y_b} \in \left[-\frac{Y_b}{2};+\frac{Y_b}{2}\right],$$

$$\xi_{Z_r} \in \left[-\frac{Z_r}{2};+\frac{Z_r}{2}\right],\ \xi_{Z_g} \in \left[-\frac{Z_g}{2};+\frac{Z_g}{2}\right],\ \xi_{Z_b} \in \left[-\frac{Z_b}{2};+\frac{Z_b}{2}\right]$$

— равномерно распределенные случайные величины; Δ — параметр технологического разброса, обусловленного качеством производства одного и того же изготовителя. Модель (18) определяет равномерный разброс допусков на значения коэффициентов X_r, X_g, X_b, Y_r, Y_g, Y_b, Z_r, Z_g, Z_b профиля ЖК панели, возникающий в процессе ее изготовления с использованием полупроводниковых светосинтезирующих элементов.

На графике цветового пространства XY технологический разброс значений коэффициентов X_r, X_g, X_b, Y_r, Y_g, Y_b, Z_r, Z_g, Z_b может быть представлен семейством треугольников, совмещенных друг с другом. Эти треугольники в совокупности образуют геометрическое место точек, воспроизводимое множеством образцов серийно выпускаемых экранов одного и того же изготовителя. Геометрическое место точек множества треугольников цветового охвата на XY-плоскости приведено на рис.1. График на рис. 1 получен методом математического моделирования выражения (18). Совмещено 10^5 треугольников цветового охвата. Для определенности в модели (18) при моделировании параметр Δ принят равным 0,1.

Для получения рис. 1 была написана специализированная программа в среде моделирования Mathcad 15.0. Исходными данными для программы являются:

- коэффициенты профиля ЖК панели, заданные для определенности модели (18) значениями $X_r = 0{,}478$; $X_g = 0{,}299$; $X_b = 0{,}175$; $Y_r = 0{,}263$, $Y_g = 0{,}650$; $Y_b = 0{,}081$; $Z_r = 0{,}020$; $Z_g = 0{,}160$; $Z_b = 0{,}908$;
- компоненты кода $RGB = (255,0,0)$, $RGB = (0,255,0)$, $RGB = (0,0,255)$, определяющие значения координат цветности вершин (x_R, y_R), (x_G, y_G), (x_B, y_B) треугольника цветового охвата в красном, зеленом и синем цветах.

Расчет координат цветности вершин треугольника цветового охвата выполнен по следующим формулам:

$$x_R = \frac{X_r}{X_r+Y_r+Z_r},\ y_R = \frac{Y_r}{X_r+Y_r+Z_r},\ x_G = \frac{X_g}{X_g+Y_g+Z_g},$$

$$y_G = \frac{Y_g}{X_g+Y_g+Z_g},\ x_B = \frac{X_b}{X_b+Y_b+Z_b},\ y_B = \frac{Y_b}{X_b+Y_b+Z_b}.$$

Моделирование случайных величин ξ_{X_r}, ξ_{X_g}, ξ_{X_b}, ξ_{Y_r}, ξ_{Y_g}, ξ_{Y_b}, ξ_{Z_r}, ξ_{Z_g}, ξ_{Z_b}, распределенных

Технологический разброс значений (*x*, *y*)-координат цветности вершин треугольника цветового охвата ЖК панелей различных изготовителей

Координата	LG-Philips, модель LM151X2 (CCFL)				Samsung, модель LTN154X1-L02 (CCFL)			
	min	Среднее значение	max	Экспертная оценка Δ	min	Среднее значение	max	Экспертная оценка Δ
x_R	0,6	0,63	0,66	0,1	0,550	0,580	0,610	0,1
y_R	0,31	0,34	0,37		0,310	0,340	0,370	
x_G	0,27	0,30	0,33		0,280	0,310	0,340	
y_G	0,57	0,60	0,63		0,520	0,550	0,580	
x_B	0,11	0,14	0,17		0,125	0,155	0,185	
y_B	0,07	0,10	0,13		0,125	0,155	0,185	
x_W	0,29	0,32	0,35		0,283	0,313	0,343	
y_W	0,31	0,34	0,37		0,299	0,329	0,359	
Координата	AU Optronics Corporation, модель G104SN03 V.0 (CCFL)				Emerging Technology Corporation, модель ETMV570G2DHU (LED)			
x_R	0,540	0,570	0,600	0,1	0,56	0,61	0,66	0,2
y_R	0,290	0,320	0,350		0,31	0,36	0,41	
x_G	0,270	0,300	0,330		0,28	0,33	0,38	
y_G	0,530	0,560	0,590		0,51	0,56	0,61	
x_B	0,115	0,145	0,175		0,09	0,14	0,19	
y_B	0,100	0,130	0,160		0,07	0,12	0,17	
x_W	0,280	0,320	0,340		0,26	0,31	0,36	
y_W	0,300	0,330	0,360		0,30	0,35	0,40	
Координата	Chi Mei Optoelectronics Corporation, модель R190E1-L01 (CCFL)				Starry Electronics Technology Corp., модель 20811010210006 (LED)			
x_R	0,618	0,648	0,678	0,1	0,55	0,58	0,61	0,1
y_R	0,303	0,333	0,363		0,32	0,35	0,38	
x_G	0,254	0,284	0,314		0,29	0,32	0,35	
y_G	0,582	0,612	0,642		0,56	0,59	0,62	
x_B	0,120	0,150	0,180		0,12	0,15	0,18	
y_B	0,045	0,075	0,105		0,09	0,12	0,15	
x_W	0,310	0,313	0,316		0,24	0,28	0,32	
y_W	0,326	0,329	0,332		0,29	0,33	0,37	
Координата	Gi Far Technology Corporation, модель GFT070BB800480-DL (LED)				Optrex Corporation, модель T-51638D084J-FW-A-AB (CCFL)			
x_R	0,550	0,580	0,610	0,1	0,528	0,558	0,588	0,1
y_R	0,284	0,314	0,340		0,297	0,327	0,357	
x_G	0,271	0,301	0,331		0,288	0,318	0,348	
y_G	0,524	0,564	0,594		0,494	0,524	0,554	
x_B	0,118	0,148	0,178		0,125	0,155	0,185	
y_B	0,093	0,123	0,153		0,108	0,138	0,168	
x_W	0,283	0,313	0,343		0,283	0,313	0,343	
y_W	0,299	0,329	0,359		0,299	0,329	0,359	

Примечание: CCFL — Cold Cathode Fluorescent Lamp; LED — Light Emitting Device.

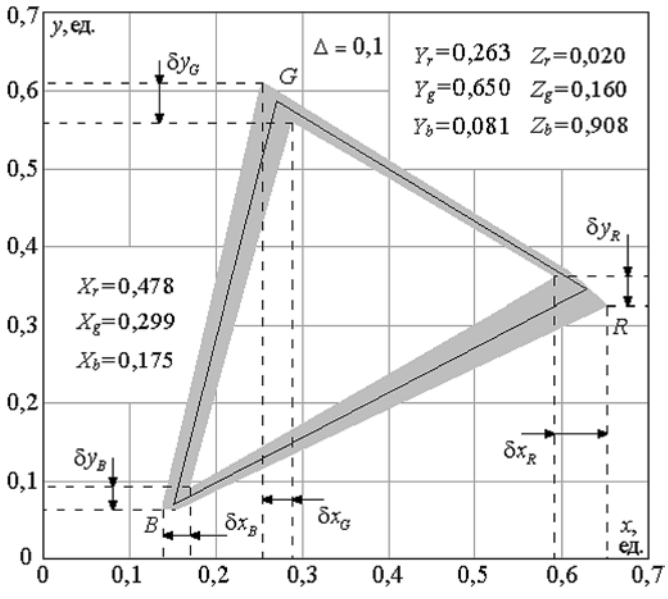

Рис. 1. Геометрическое место точек множества треугольников цветового охвата на *XY*-плоскости

по равномерному закону, проводилось с использованием встроенного датчика среды. Моделирование осуществлялось на персональном компьютере ASUS K56CB-X0391H со следующими характеристиками: процессор Intel(R) Core(TM) i5-3337U, 4 ядра, тактовая частота 1,8 ГГц, оперативная память 6 Гбайт под управлением операционной системы Windows 8.1.

Как показывает анализ рис. 1, геометрическое место точек образовано семейством треугольников цветового охвата, вершины (x_R, y_R), (x_G, y_G), (x_B, y_B) которых распределены, соответственно, в интервалах $(\delta x_R, \delta y_R)$, $(\delta x_G, \delta y_G)$, $(\delta x_B, \delta y_B)$. Сплошной черной линией на рис. 1 показан треугольник, соответствующий средним значениям вершин треугольника (в формуле (18) для $\Delta = 0$).

Графическое представление технологического разброса значений вершин множества треугольников цветового охвата модели (18) адекватно существующему на практике у изготовителей ЖК панелей разбросу.

Согласно технической документации на ЖК панели (см. таблицу), повторяемость изготовления образцов экранов гарантируется производителем в пределах $0{,}1 \leqslant \Delta \leqslant 0{,}2$. Экспертная оценка Δ в таблице дана на основе анализа рис. 1 для представленных в таблице минимально и максимально допустимых значений разброса (x, y)-координат цветности вершин треугольника цветового охвата. Не трудно видеть, что разброс координат цветности вершин треугольника для ЖК панелей с газоразрядным подсветом на основе ламп (CCFL) и с подсветом на основе светодиодов (LED) находится на одном уровне.

Значение параметра разброса Δ в модели (18) для каждой ЖК панели соответствует удвоенному значению разброса δx_R, δy_R, δx_G, δy_G, δx_B, δy_B координат цветности вершин треугольника, определенному изготовителем в документации, т. е.

$$\Delta \approx 2\delta x_R = 2\left(\max(x_R) - \min(x_R)\right) = 2\delta y_R =$$
$$= 2\left(\max(y_R) - \min(y_R)\right) = 2\delta x_G =$$
$$= 2\left(\max(x_G) - \min(x_G)\right) = 2\delta y_G =$$
$$= 2\left(\max(y_G) - \min(y_G)\right) =$$
$$= 2\delta x_B = 2\left(\max(x_B) - \min(x_B)\right) =$$
$$= 2\delta y_B = 2\left(\max(y_B) - \min(y_B)\right).$$

4. Исследование распределения координат цветности ЖК панелей одного производителя

Для исследования распределения координат цветности ЖК панелей одного производителя была написана специальная программа в среде моделирования Mathcad 15.0. Исходными данными для программы являлись:

- коэффициенты профиля ЖК панели, заданные для определенности модели (18) значениями $X_r = 0{,}478$; $X_g = 0{,}299$; $X_b = 0{,}175$; $Y_r = 0{,}263$, $Y_g = 0{,}650$; $Y_b = 0{,}081$; $Z_r = 0{,}020$; $Z_g = 0{,}160$; $Z_b = 0{,}908$;
- математическая модель (18) с уравнениями прямого преобразования Грассмана для расчета координат цвета и координат цветности;
- последовательность равномерно распределенных псевдослучайных величин ξ_{X_r}, ξ_{X_g}, ξ_{X_b}, ξ_{Y_r}, ξ_{Y_g}, ξ_{Y_b}, ξ_{Z_r}, ξ_{Z_g}, ξ_{Z_b};
- компоненты кода *RGB*, определяющие точку *XY*-плоскости, в окрестностях которой исследуется распределение координат цветности.

В процессе проведения серии модельных экспериментов установлено, что значения (x, y)-координат цветности изображения, индицируемого в любом цвете на ЖК панели одного и того же производителя, являются взаимозависимыми. Горизонтальное сечение двумерной гистограммы распределения $p(x, y)$ координат цветности на уровне порога L имеет вид эллипса, размер полуосей которого зависит от параметра технологического разброса Δ и уровня доверительной вероятности P_D.

Угол наклона главной полуоси эллипса в прямоугольной системе координат *XY*-плоскости различен для разных положений точки центра эллипса и определяется коэффициентом корреляции величин x, y. Гистограмма распределения $p(x,y)$ в программе моделирования представлена двумерным массивом относительных частот попадания координат цветности в подинтервал построения $p(x,y)$.

Уровень доверительной вероятности $P_D \in [0; 1)$ определяет относительную долю (от 0, что соответствует 0 %, до 1 — 100 %) числа (x, y)-координат цветности, попавших внутрь области, ограничиваемой эллипсом, от общего числа точек, полученных по модели (18) для фиксированного значения компонент кода *RGB* и параметра Δ.

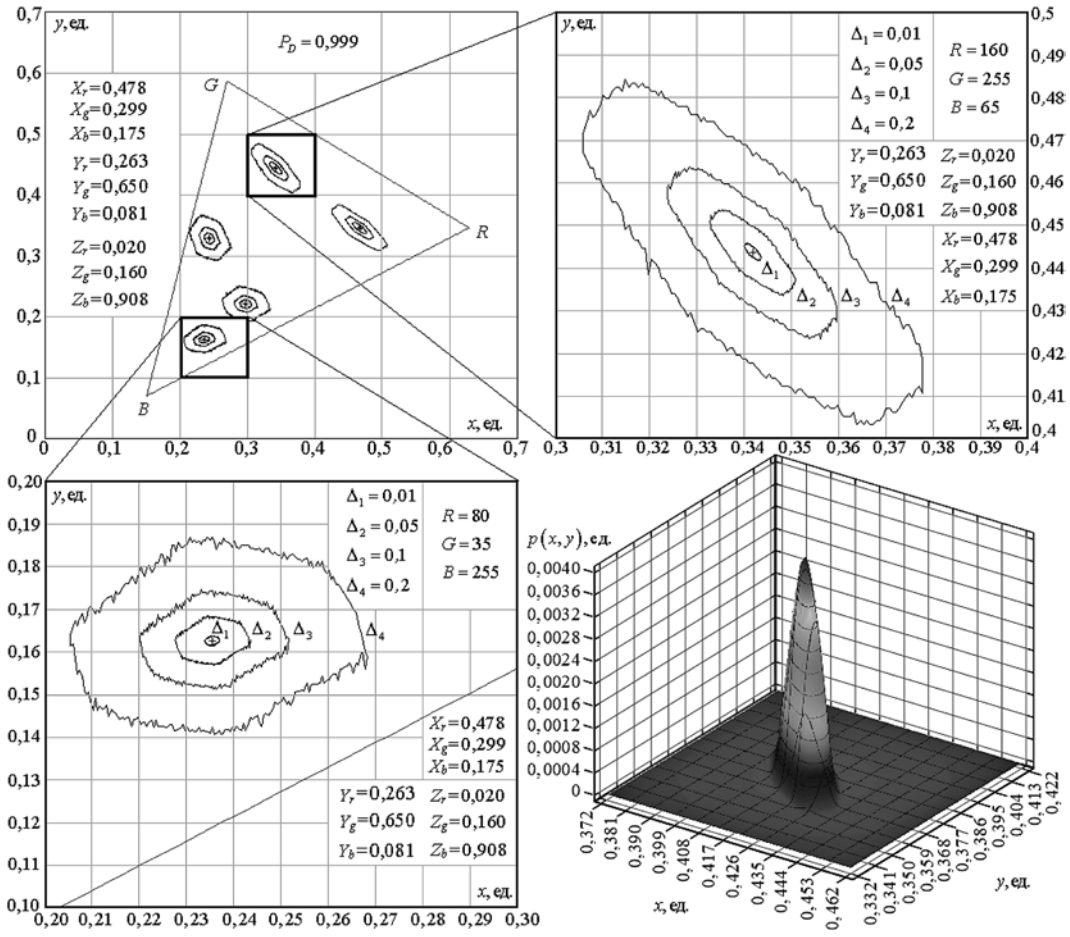

Рис. 2. Горизонтальные сечения гистограммы $p(x, y)$ и пример двумерной гистограммы распределения $p(x, y)$ для координат цветности на XY-плоскости (уровень доверительной вероятности сечений P_D = 0,999)

На рис. 2 показаны семейства сечений (эллипсов) двумерной гистограммы распределения $p(x, y)$ для различных значений параметра технологического разброса $\Delta_4 = 0{,}2$; $\Delta_3 = 0{,}1$; $\Delta_2 = 0{,}05$; $\Delta_1 = 0{,}01$. Точке центра эллипса, единой для всех Δ, соответствует точка (отмечена на графике мнемознаком "+") с (x,y)-координатами, рассчитанными по модели (18) для $\Delta = 0$.

Уровень порога L для горизонтального сечения гистограммы $p(x,y)$ определяется из уравнения

$$L = \arg\min_{L}\left(\left|\sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1} I\left(H_{k_1,k_2} - L\right) H_{k_1,k_2} - P_D\right|\right), \quad I(q) = \begin{cases} 0, q < 0 \\ 1, q \geqslant 0 \end{cases}, \qquad (19)$$

где H_{k_1,k_2} — относительные частоты попадания (x, y)-координат цветности в подинтервал построения гистограммы $p(x,y)$; $I(q)$ — индикаторная функция; N — число подинтервалов гистограммы $p(x, y)$ по каждой из координат.

Уравнение (19) в программе моделирования решено относительно уровня порога L численным методом. Размер подинтервала построения гистограммы $p(x, y)$ на XY-плоскости составил 0,0003×0,0003 ед.

Горизонтальные сечения гистограммы $p(x, y)$ на рис.2 показаны для уровня доверительной вероятности P_D = 0,999 в различных областях треугольника цветового охвата для пяти различных ненулевых значений кода *RGB*.

Для $\Delta = 0{,}2$ площадь эллипса включает порядка $5 \cdot 10^5$ значений (x, y)-координат цветности, определенных с учетом разрешающей способности [5] преобразования Грассмана. Вершины треугольника цветового охвата получены из модели (18) при $\Delta = 0$, т. е. по классификации (см. таблицу) соответствуют среднему значению координат цветности (x_R, y_R), (x_G, y_G), (x_B, y_B).

Сечения гистограммы $p(x, y)$ на различных уровнях порога L с достаточной для практики точностью соответствуют сечению, которое получается по теоретической двумерной функции плотности вероятности для зависимых нормально распределенных случайных величин x, y:

$$f(x,y)=\frac{1}{2\pi\sqrt{D_xD_y}\sqrt{1-\rho^2}}\times$$
$$\times\exp\left(-\frac{1}{2\left(1-\rho^2\right)}\left(\frac{\left(x-M_x\right)^2}{D_x}-\frac{2\rho\left(x-M_x\right)\left(y-M_y\right)}{\sqrt{D_xD_y}}+\frac{\left(y-M_y\right)^2}{D_y}\right)\right), \quad (20)$$
$$\rho=\frac{\frac{1}{N}\sum_{i=0}^{N-1}\left(x_i-M_x\right)\left(y_i-M_y\right)}{\sqrt{D_xD_y}},$$

где ρ — коэффициент корреляции случайных величин x, y; M_x, M_y — математические ожидания случайных величин x, y соответственно; D_x, D_y — дисперсии случайных величин x, y соответственно. В формуле (20) параметры M_x, M_y, D_x, D_y, ρ могут быть заменены соответствующими выборочными оценками.

Уравнение семейства эллипсов равных плотностей вероятности, образованное горизонтальным сечением теоретической функции (20) на уровне порога λ, имеет следующий вид [7]:

$$Q(x,y)=$$
$$=\frac{\left(x-M_x\right)^2}{D_x}-\frac{2\rho\left(x-M_x\right)\left(y-M_y\right)}{\sqrt{D_xD_y}}+.$$
$$+\frac{\left(y-M_y\right)^2}{D_y}=\lambda^2.$$

Разным значениям порога λ соответствуют разные, но постоянные вероятности для всех точек эллипса при заданной функции (20). Вероятность того, что точка функции плотности вероятности (20) со случайными (x, y)-координатами цветности, распределенными по двумерному нормальному закону, окажется внутри эллипса равных вероятностей с фиксированным порогом λ, рассчитывается следующим образом [7]:

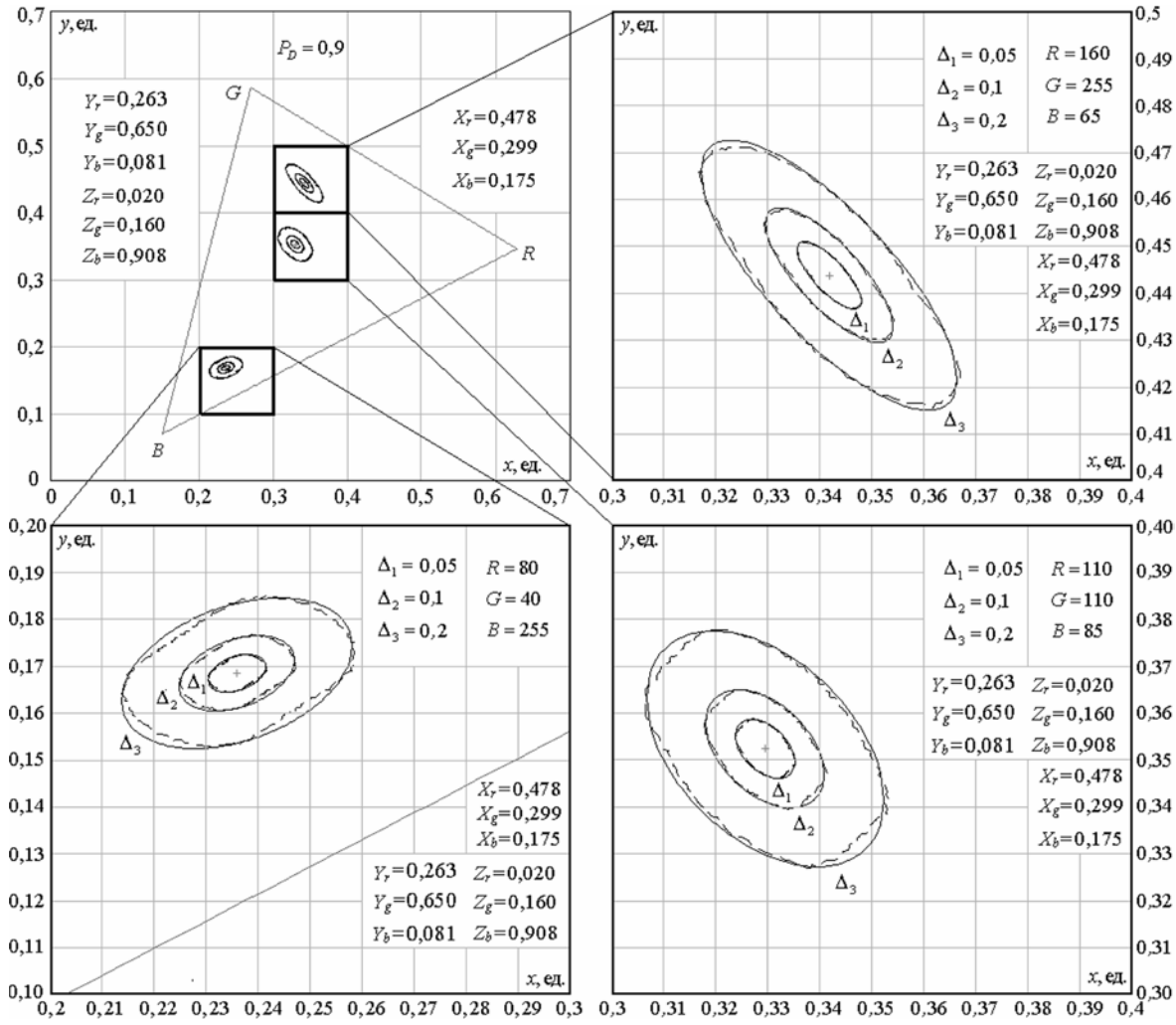

Рис. 3. Семейство эллипсов сечений гистограммы $p(x, y)$ (штриховая линия) и функции $f(x, y)$ плотности вероятности (сплошная линия)

$$P_D(\lambda) = \frac{1}{2\pi\sqrt{D_x D_y}\sqrt{1-\rho^2}} \times$$

$$\times \iint\limits_{g(\lambda)} \exp\left(-\frac{1}{2\left(1-\rho^2\right)} Q(x,y)\right) dxdy = \qquad (21)$$

$$= 1 - \exp\left(-\frac{\lambda^2}{2\left(1-\rho^2\right)}\right),$$

где $g(\lambda)$ — область определения плотности распределения (x, y), ограниченная эллипсом. Решение уравнения (21) относительно λ имеет следующий вид:

$$\lambda^2 = -2\left(1-\rho^2\right)\ln\left(1-P_D\right). \qquad (22)$$

Семейство эллипсов сечений функции $f(x, y)$ на XY-плоскости изображено на рис. 3. Уровень порога λ определен по формуле (22) для доверительной вероятности $P_D = 0{,}9$. Гипотеза о нормальности двумерного распределения координат цветности (x, y) может быть подтверждена проверкой по критерию согласия Колмогорова-Смирнова, обобщенному в работе [8] на многомерный случай.

Анализ рис. 3 показывает, что разброс (x, y)-координат цветности, обусловленный технологическим разбросом параметров изготовления ЖК панелей, оказывается существенным даже при $\Delta = 0{,}01$: разброс x-координаты находится в пределах 0,005 ед.; разброс y-координаты находится в пределах 0,005 ед. Как видно по данным таблицы, производители экранов в настоящее время обеспечивают повторяемость изготовления на уровне разброса $0{,}1 \leqslant \Delta \leqslant 0{,}2$.

Разброс 0,005 ед. при $\Delta = 0{,}01$ чувствителен для наблюдателя, он может быть измерен современными средствами прямых измерений (колориметрами) и вполне допустим для качества исполнения аппаратуры бытового применения (видеомониторы, телевизоры, телефоны, ipad и др.), когда наблюдателю одновременно предъявлен один единственный образец средства отображения. Однако значения $\Delta = 0{,}01$ недостаточно для изготовления ЖК панелей для МФЦИ авиационного применения, когда наблюдатель воспринимает изображение одновременно на нескольких (до 6 шт.) бортовых индикаторах, установленных на приборную панель самолета.

Заключение

Разброс значений (x, y)-координат цветности индицируемого изображения, обусловленный технологическим разбросом параметров изготовления ЖК панелей, с достаточной для практики точностью описывается двумерным нормальным законом распределения для зависимых случайных величин.

Значение коэффициента корреляции, определяющее угол наклона главной полуоси эллипса (см. рис. 2 и 3), не является постоянной величиной в пределах всего треугольника цветового охвата на XY-плоскости и зависит от значений компонент кода *RGB*, т. е. от точки, (x, y)-координаты которой соответствуют точке центра эллипса при $\Delta = 0$ в модели (18).

Выражения (1)—(4), определяющие правило взаимного соответствия координат цвета (цветности) элементов изображения, индицируемых на ЖК панелях различных производителей, приведены в общем виде и могут использоваться для оценки свойств цветопередачи экранов любых производителей. При этом расчет коэффициентов профиля ЖК панели необходимо осуществлять по формулам (9)—(17).

Разброс координат цветности ЖК панелей одного производителя может быть компенсирован преобразованием (4) за счет приведения колориметрических характеристик цветопередачи экранов группы МФЦИ одного самолета к колориметрическим характеристикам цветопередачи одного (эталонного) экрана, профиль которого соответствует, например, средним значениям вершин треугольника цветового охвата. Выравнивание колориметрических характеристик цветопередачи группы экранов осуществляется путем подстановки в формулу (4) коэффициентов профиля ЖК панели, рассчитанных по формулам (7)—(19) для каждого образца ЖК панели после светотехнических измерений координат цветности в красном, зеленом, синем и белом цветах.

Выражения (1)—(4) справедливы также для описания процессов цветопередачи в полиграфической промышленности, компьютерном дизайне и других областях, в которых колориметрическое описание изображения формируется с одним профилем средства формирования изображения, например, с помощью фотоаппарата, и воспроизводится на других средствах индикации, например, на мониторах или принтерах с отличающимися профилями.

Список литературы

1. **Жаринов И. О., Жаринов О. О.** Бортовые средства отображения информации на плоских жидкокристаллических панелях: учеб. пособие. Информационно-управляющие системы, СПб.: ГУАП, 2005. 144 с.

2. **Barber S. et al.** US Patent 7,417,641 B1: Aeronautical chart display apparatus and method, Aug. 26. 2008.

3. **Костишин М. О., Жаринов И. О., Жаринов О. О.** Исследование визуальных характеристик средств отображения пилотажно-навигационных параметров и геоинформационных данных в авионике // Информационно-управляющие системы. 2014. № 4. С. 61—67.

4. **Menesatti P., Angelini C., Pallottino F., Antonucci F., Aguzzi Y., Costa C.** RGB color calibration for quantitative image analysis: the "3D thin-plate alpine" warping approach // Sensors. 2012. N. 12. P. 7063—7079.

5. **Жаринов И. О., Жаринов О. О.** Исследование распределения оценки разрешающей способности преобразования Грассмана в системах кодирования цвета, применяемых в авионике // Программная инженерия. 2014. № 8. С. 40—47.

6. **Zargaryants G. S., Mikhailov O. M.** Integral remote colorimeter bases on the RGB colorimetric system // Light & Engineering. 2008. Vol. 16, N. 3. P. 69—77.

7. **Левин Б. Р.** Теоретические основы статистической радиотехники. М.: Сов. радио, 1974, кн. 1, 522 с.

8. **Justel A., Pena D., Zamar R.** A multivariate Kolmogorov—Smirnov test of goodness of fit //Statistics & Probability Letter. 1997. Vol. 35, N. 3. P. 251—259.

I. O. Zharinov, Associate Professor, Chief of Department, Saint Petersburg National Research University of Information Technologies, Mechanics and Optics (University ITMO), Chief of Learning-Scientists Center, SPb Scientific Design Bureau "Electroavtomatica" n. a. P. A. Efimov, e-mail: igor_rabota@pisem.net,
O. O. Zharinov, Associate Professor, Saint-Petersburg State University of Aerospace Equipment

The Research of the Mathematical Model of Color Representation of LCD-panels

The mathematical model of color representation of LCD-panels is researched. The mathematical expressions describing the shifts of chromaticity coordinates for different samples of LCD panels by various manufacturers and by one the same manufacturer are presented. The mathematical expressions for calculation of the profile coefficients for any LCD-panel through the given values of coordinates of the chromatic triangle vertexes and white color point are presented. The results, obtained by mathematical simulation technique, are presented, demonstrating a possible use of two-dimensional normal distribution to describe chromaticity coordinates, which are random because of manufacturer's technological variations.

Keywords: *color coordinates, chromaticity coordinates, indication systems, model of color represntation, LCD-panel*

References

1. **Zharinov I. O., Zharinov O. O.** *Bortovye sredstva otobrazhenija informacii na ploskih zhidkokristallicheskih paneljah* (Onboard Display on Flat Liquid Crystal Panels), Saint-Petersburg, GUAP, 2005, 144 p. (in Russian).

2. **Barber S.** US Patent 7,417,641 B1: Aeronautical chart display apparatus and method, Aug. 26. 2008.

3. **Kostishin M. O., Zharinov I. O., Zharinov O. O.** Issledovanie vizual'nykh kharakteristik sredstv otobrazheniya pilotazhno-navigatsionnykh parametrov i geoinformatsionnykh dannykh v avionike. *Informacionno-upravljajushhie sistemy*, 2014, no. 4, pp. 61—67 (in Russian).

4. **Menesatti P., Angelini C., Pallottino F., Antonucci F., Aguzzi Y., Costa C.** RGB color calibration for quantitative image analysis: the "3D thin-plate alpine" warping approach. *Sensors*, 2012, no. 12, pp. 7063—7079.

5. **Zharinov I. O., Zharinov O. O.** Issledovanie raspredeleniya otsenki razreshayushchei sposobnosti preobrazovaniya Grassmana v sistemakh kodirovaniya tsveta, primenyaemykh v avionike. *Programmnaja ingeneria*, 2014, no. 8, pp. 40—47 (in Russian).

6. **Zargaryants G. S., Mikhailov O. M.** Integral remote colorimeter bases on the RGB colorimetric system. *Light & Engineering*, 2008, vol. 16, no. 3, pp. 69—77.

7. **Levin B. R.** *Teoreticheskie osnovy statisticheskoj radiotehniki* (Theoretical foundations of statistical radio engineering), Moscow: Sov. radio, 1974, P. 1, 522 p. (in Russian).

8. **Justel A., Pena D., Zamar R.** A multivariate Kolmogorov-Smirnov test of goodness of fit. *Statistics & Probability Letter*, 1997, vol. 35, no. 3, pp. 251—259.

ИНФОРМАЦИЯ

УДК 004.9

Н. А. Маркова, канд. физ.-мат. наук, вед. науч. сотр., e-mail: MarkovaNatAlex@gmail.com,
Институт проблем информатики ФИЦ ИУ РАН, Москва

Технология поддержки конкретно-исторических исследований на основе модели фактоподобных высказываний

Исходя из специфики конкретно-исторических исследований, разработан метод формализованного представления данных, учитывающий их неточность и темпоральные свойства. На базе этого метода создана информационная технология поддержки работы исследователей.

Ключевые слова: *конкретно-историческое исследование, объектная модель, фактоподобное высказывание, информационная технология*

Введение

Принимающие в настоящее время массовый характер электронные публикации источников исследований и их результатов открывают дополнительные возможности для изучения истории конкретных лиц, организаций, сфер деятельности. Для того чтобы работа исследователя на этом направлении была эффективной, необходимо представлять исторические сведения — факты в удобном для использования машиночитаемом виде.

Ряд особенностей конкретно-исторических исследований не позволяет применить унифицированные готовые решения, опирающиеся на представление о факте, как об утверждении, что, в частности, неявно предполагает использование технологии семантического веба [1]. Далеко не все факты, излагаемые в исторических источниках, в основанных на них исследованиях, а также в справочниках и энциклопедиях, соответствуют объективной истине. Документы нередко содержат предположения, гипотезы, частичное знание об интересующем предмете, не говоря уже о вольных или невольных искажениях сведений о нем. Приближение к истине возможно за счет анализа противоречий, интеграции данных, которые извлекаются из различных источников. Модели, используемые для представления фактографической информации (например, в проекте dbPedia [2]), не учитывают чрезвычайно важные для конкретно-исторических исследований особенности, а именно темпоральность и неточность.

Расширим понятие "факт", включив в него неточные и неполные сведения, результаты их аналитико-синтетической обработки, вопросы и гипотезы. Предложим общую форму для фиксации такого рода сведений в виде фактоподобных высказываний. Представляемая модель объединяет: данные, извлекаемые из источников; результаты их анализа и обработки; сведения, позволяющие отслеживать процесс исследования. Основные положения предлагаемого в настоящей работе формализма будем выражать в терминах объектной модели, совпадающих на концептуальном уровне с категориями аппарата онтологий. Такой аппарат повсеместно применяется в настоящее время для формального представления фактических знаний. Модель основана на анализе специфики конкретно-исторических исследований. Цель ее построения — создание основы для разработки эффективной информационной технологии поддержки работы исследователей.

1. Конкретно-исторические исследования

Примерами конкретно-исторических исследований являются: изучение биографии конкретного лица; истории организации или сферы деятельности; краеведение, генеалогия. Многие научные работы из самых разных областей начинаются с обзора истории изучения исследуемой проблемы. В рамках такого рода исследований изучаются объекты исторической реальности и их свойства. Набор классов изучаемых объектов зависит от вида исследований, однако по крайней мере два класса характерны для любых проектов — это люди и документы, в которых фиксированы сведения об изучаемой реальности.

В любом случае при организации хранения формализованных конкретно-исторических сведений приходится учитывать следующие особенности:

- вариативный характер и неоднозначность имен объектов и значений свойств;
- ограничения времени существования объекта и периодов постоянства значений большинства его свойств;
- зависимость перечня (набора) свойств объекта от его класса и конкретного исследования;

- наличие искажений, дефектов в источниках, ошибок при их распознавании, интерпретации, интеграции.

Причин искажений много. Это и случайные описки, и преднамеренная фальсификация, плохая сохранность документа, неразборчивый почерк и другие причины. Как утверждал историк Л. Н. Гумилев в своих популярных лекциях: "Источники все врут". Тем не менее и с плохими данными при их анализе, сопоставлении и интеграции можно работать.

Для того чтобы облегчить исследователю работу с конкретно-исторической информацией, необходимо создать информационную технологию, позволяющую:

- фиксировать не только четко установленные факты, но и фактоподобные высказывания, включая предположения, неточные значения, вопросы;
- соотносить высказывание с источником/цепочкой вывода;
- обеспечить навигацию по связям, провести многоаспектный поиск, предоставить возможность статистической обработки;
- отслеживать процессы накопления данных, выявления дефектов, выдвижения/опровержения гипотез по источникам, времени, исполнителям, аргументации.

Основой для создания такой технологии должна стать модель представления фактоподобных высказываний и процессов работы с ними.

2. Модель фактоподобных высказываний

Большинство из изучаемых в конкретно-исторических исследованиях объектов в какой-то мере может быть описано при помощи современных моделей связанных данных [3], которых на настоящее время насчитывается сотни. В них подробно прописаны свойства лиц, организаций, географических объектов и т. д. К сожалению, данные, которые фиксируются моделями, обладают темпоральной, модальной и процессной неопределенностью. В частности, нельзя фиксировать степень достоверности факта, а также его отношение к конкретным источникам.

Проиллюстрируем суть возникающих трудностей на примере центральной модели связанных данных — dbPedia. В статье, посвященной А. П. Чехову (http://dbpedia.org/page/Anton_Chekhov), фиксируется, что он закончил Московский университет (Category:Moscow_State_University_alumni), но ответа на вопрос "когда?" там нет. Тем самым, используя базу знаний dbPedia, нам не удастся, например, ответить на вопрос, кто учился вместе с Чеховым. В исходной статье Wikipedia данные о времени обучения были. На поставленный вопрос удалось бы ответить, если бы факт обучения включал временной диапазон.

Представим модель фиксации конкретно-исторических сведений, учитывающую их специфику. Модель является развитием решений, предложенных в работах [4, 5]. Ее составляют формализованные высказывания, представляющие:

- модель области исследований (μ-факты);
- объекты исследования и сопряженные с ними сущности (ξ-факты);
- процесс исследований (φ-факты).

В целом, будем рассматривать π-факты — фактоподобные высказывания. Определим их конструкцию, используя теоретико-множественный формализм, начиная с ξ-фактов.

Для того чтобы отразить реалии исследовательской работы, необходимо давать оценку правдоподобия высказывания, включающую не только значения ПРАВДА (*T*) и ЛОЖЬ (*F*), но и промежуточные значения. В качестве таких значений могут быть выбраны как диапазон нечеткой логики [0-1], так и номинальная шкала. На практике вполне достаточно включить в такую шкалу промежуточные значения: "скорее ЛОЖЬ" (?*F*), "скорее ПРАВДА" (?*T*), "равновероятно ПРАВДА и ЛОЖЬ" (?). Эти значения фиксируют как степень сомнения в правдивости утверждения, так и вопросы или предположения, формулируемые исследователем. Будем ставить в соответствие высказываниям оценку их правдоподобия в терминах шкалы $E = \{T, ?T, ?, ?F, F\}$.

Пусть $O = \cup O_{class}$ — объекты исследования, где O_{class} — объекты класса *class*. Говоря о конкретном объекте $o \in O$, будем использовать нижние индексы, отсылающие к его дефиниции — d, d_p, d_q.

Свойства объектов будем рассматривать во времени *t*. Обозначим $\mathrm{d}t = (start, finish)$ — временной промежуток, границы которого могут быть определены строго, либо заданы допустимыми значениями. Подробно варианты фиксации временных промежутков рассмотрены в работе [5].

Будем рассматривать литеральные свойства (*a*) и отношения объектов (*r*), конкретный вид которых уточняется индексом (*aclass* и *rclass* соответственно). Значения свойств будем обозначать *avalue* и *rvalue*.

Итак, ξ-фактами являются перечисленные далее виды высказываний.

- Дефиниция (*D*) — высказывание, фиксирующее принадлежность объекта к некоторому классу:

$$\forall t \in \mathrm{d}t\ (o_d(t) \in O_{class}).$$

- Атрибуты (*A*) — высказывание о литеральном свойстве объекта:

$$\forall t \in \mathrm{d}t\ (a_{aclass}(o_d,\ t) \bullet avalue).$$

- Отношения объектов (*R*) — высказывание об объектных свойствах объектов:

$$\forall t \in \mathrm{d}t\ (r_{rclass}(o_{dp},\ o_{dq},\ t) \bullet rvalue).$$

Здесь $\bullet \in \{=,\ \approx, \neq, <, >, \in, \notin\} \times E$ — оператор сопоставления.

- Высказывание о темпоральных отношениях между свойствами объектов (*T*):

$$p_1 \ \textcircled{t}\ p_2,$$

где $p_1, p_2 \in A \cup R$,

$\textcircled{t} \in \{<\textcircled{t}, >\textcircled{t}, =\textcircled{t}, \neq \textcircled{t}, ...\} \times E$ — сопоставление временных интервалов.

- Высказывание о логическом отношении между свойствами объектов (L):

$$p_1 \blacklozenge p_2 \blacklozenge ... \blacklozenge p_n,$$

$$p_1, p_{2, ...} p_n \in A \cup R \cup T \cup L,$$

$\blacklozenge \in \{\wedge, \vee, \wedge\neg, \vee\neg, \oplus, ...\} \times E$ — логические связки.

Множество ξ-фактов складывается из высказываний перечисленных выше видов:

$$\{\xi\} = D \cup A \cup R \cup T \cup L.$$

Представим исследовательские операции как φ-факты трех видов:

- интерпретация — сопоставление ξ-факту документального объекта d — источника, на основании которого он сформулирован ($\xi \Leftarrow d$);
- представление — сопоставление ξ-факту документального объекта d, в котором он опубликован ($\xi \Rightarrow d$);
- умозаключение — сопоставление ξ-фактам — ξ-факт-вывод ($\xi | - \xi$).

Деятельность, фиксируемая φ-фактом, неформальна. Тексты документов принципиально не формализуемы. Но и в случае умозаключений вывод из формализованных высказываний делается с привлечением экспертного знания и интуиции исследователя. Ссылка на лицо (его дефиницию), осуществляющее исследование, и время выполнения операции — компоненты φ-факта.

Набор μ-фактов — это списки определений классов объектов (*class*), классов их свойств (*aclass*, *rclass*), их возможных значений (*avalue*, *rvalue*), а также накладываемых на них ограничений.

3. Документы и фактоподобные высказывания

Безусловно, любая формализация связана с потерями. Текст документа обладает внутренним единством, вычленение конкретного факта всегда ущербно. При этом теряется контекст, эмоциональная окраска высказывания, стилистические нюансы. Формализованные данные, однако, значительно легче анализировать и интегрировать, с их помощью может быть организован эффективный поиск. Их роль — роль метаданных, сопровождающих основной массив документов, его справочно-информационный аппарат. Вместе с тем π-факты, отражающие как действия исследователя, так и сведения о предметной области, рационально сопровождать неформальным комментарием, в качестве которого может выступать не только текст, но и рисунок или произвольный мультимедийный объект. Этот комментарий может содержать цитату или изложение источника, а также отсылки на документ или конкретное место в документе, которые, в свою очередь, могут содержать отсылки на π-факты (аналогичные сноскам печатных изданий). Таким образом формируется многосвязная интертекстуальная картина, в рамках которой появляется возможность организовать навигацию между текстами и формальными сведениями, связанными с ними.

Документ-источник, публикация или черновик, и сам является объектом конкретно-исторического исследования, ему соответствует набор ξ-фактов: дефиниция, атрибуты, отношения. Своеобразными объектами исследования являются и его субъекты — персонал научного проекта, организации и лица, взаимодействующие с ним, а также имеющие отношение к используемым источникам/ целевым материалам. Концептуальную модель, эффективно представляющую объекты такого рода, создали библиографы [6]. Целесообразно фиксировать в виде ξ-фактов следующие отношения между документами:

- структурные (входит, следует за);
- эквивалентность (копия, репринт);
- деривативные (версии, издания, переработки, переводы);
- дескриптивные (критика, комментарии, аннотации, и т. п.).

Для того чтобы реализовать взаимодействие документов и π-фактов, необходимо воспользоваться принципами открытых связанных данных. Электронные документы и их части адресуются посредством универсального идентификатора (URI). В какой-то мере URI в форме ISBN работает и с книжными изданиями. Старые книги, архивные документы могут быть идентифицированы на основе совокупности идентификатора хранилища (библиотеки, архива) и используемого в данном хранилище шифра хранения. Унифицированный идентификатор ресурса π-факта будем строить как совокупность идентификатора инструментальной оболочки, адреса хранения коллекции π-фактов и уникального идентификатора π-факта в коллекции. Тогда переходы между π-фактами, а также между π-фактами и документами можно организовать одним щелчком мыши по принципу гиперссылки.

4. Информационная технология, основанная на модели фактоподобных высказываний

Для того чтобы работа исследователя была эффективной, ему необходимы:

- справочно-информационный аппарат, позволяющий оперативно сопоставить данные изучаемого источника с уже накопленным материалом;
- удобные интерфейсные формы, позволяющие фиксировать новое знание вместе с отсылками на источники или аргументы умозаключения;
- выборка данных исследования по разнообразным критериям и их наглядное представление;
- развитые формы поиска и навигации;
- алгоритмы контроля корректности и непротиворечивости данных.

Для реализации такой поддержи целесообразно представить знания конкретной предметной области,

а также сведения, характеризующие процесс исследования, в виде коллекции взаимосвязанных фактоподобных высказываний. Рассмотрим технологические решения, основанные на таком подходе, опробованные при реализации и эксплуатации инструментального комплекса Фактограф [7].

Коллекция π-фактов составляет ядро информационного поля, включающего также документы-источники, целевые материалы, промежуточные данные. Работа с компонентами информационного поля поддерживается соответствующим инструментарием — электронным (в частности, офисными приложениями) или традиционным (аппаратом архивов и библиотек). Для работы с ядром информационного поля используется специальный инструмент, обеспечивающий разнообразные формы ввода, визуализации и анализа данных.

Исследователю предоставляются в той или иной степени автоматизированные формы ввода, учитывающие особенности конкретного исследования. В интерфейсной форме, содержащей данные об объекте, в соответствии с его классом выделяются разделы, представляющие различные группы свойств. Например, для лица — это связи с организациями (службы/обучения), родство, социальные статусы (рис. 1).

Удобной формой представления структурных связей является иерархия. С ее помощью визуализируются (и редактируются) родословные древа, организационные структуры, а также структуры вложенности объектов произвольной природы, в частности, географических и документальных (рис. 2). Как правило, речь идет о нестрогой иерархии. Для документальных объектов кроме "естественной" вложенности (например, архив-фонд-опись-дело-лист),

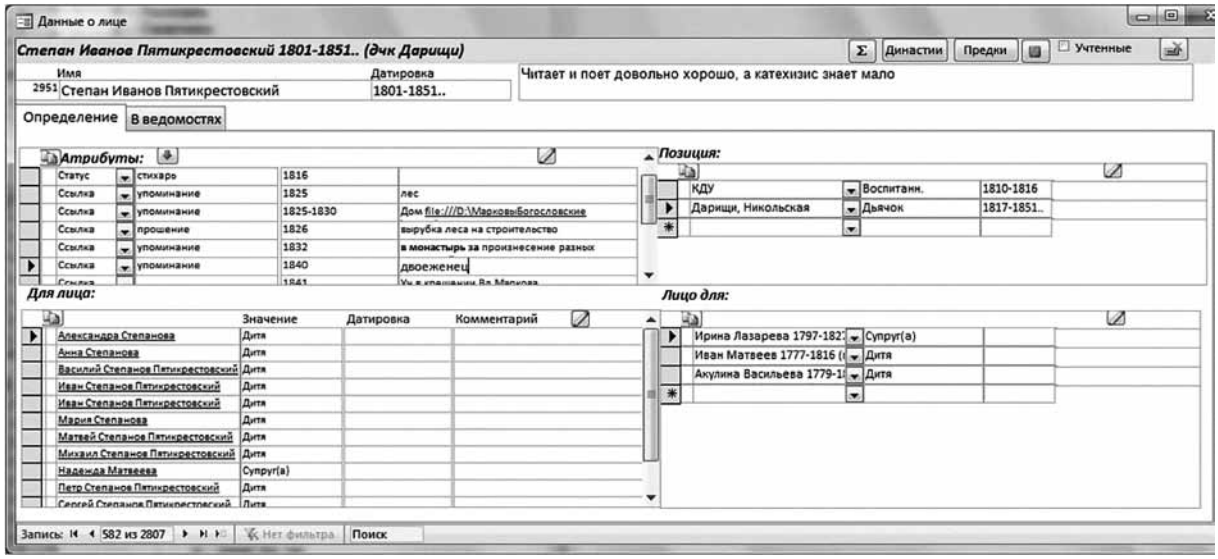

Рис. 1. Интерфейсная форма данных о лице

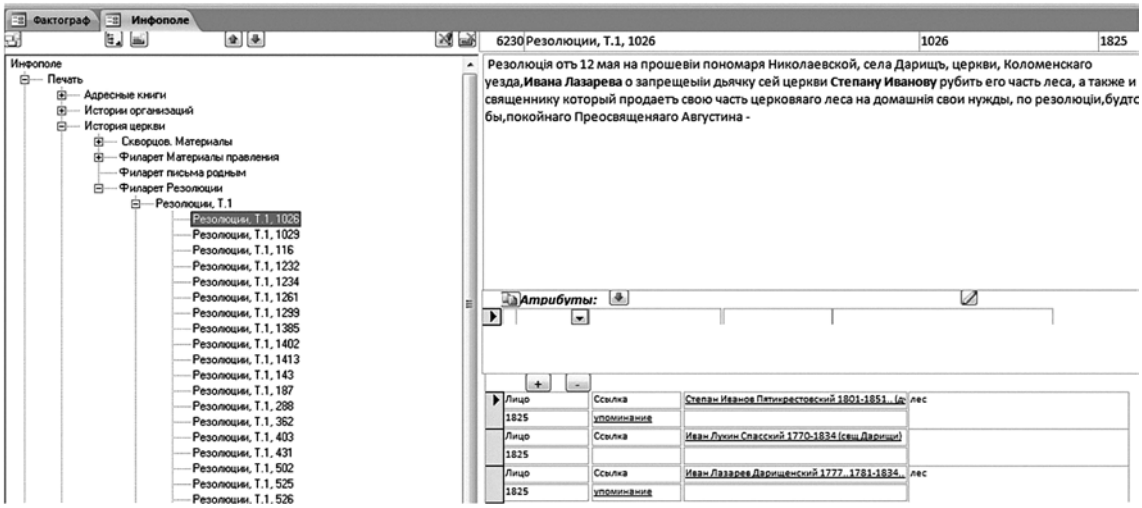

Рис. 2. Иерархия документальных объектов

исходя из логики конкретной работы, исследователем вводятся дополнительные деревья, например, "коллекций": письма — письма с фронта — письма медперсонала. Вложенность географических объектов также неоднозначна. Один и тот же населенный пункт в разное время мог принадлежать разным административным единицам.

В зависимости от вида исследования данные могут быть сгруппированы и в другие визуальные структуры: списки; плоские и сводные таблицы; различные формы инфографики.

Учитывая естественный, соответствующий реалиям предметной области и процесса исследования принцип формулировки π-фактов, оперативная навигация по их связям является эффективным способом перехода между взаимосвязанными сущностями.

Поиск по текстовым шаблонам в рамках предлагаемой технологии приобретает новые черты. Во-первых, поиск внутри коллекции π-фактов сочетает возможности атрибутного (фасетного) и полнотекстового подходов. Поисковый образ вместе с текстовым шаблоном может включать набор ограничений на значения атрибутов и/или отношений объектов.

Во-вторых, опираясь на фиксированные атрибуты и связи объектов, можно формировать запросы, текст которых далек от идентифицирующих данных. Например, если некое лицо, "Павлов А. И.", отмечен как дядя автора воспоминаний, то сведения о нем следует искать, опираясь на поисковый образ "дядя Леша". В письмах воспитанников возглавляемой им гимназии он будет фигурировать как "Директор".

Большинство из алгоритмов контроля конкретно-исторических данных специфично для конкретного вида исследований. Фактически для любого исследования актуален контроль, связанный с проверкой временных диапазонов: события жизни человека располагаются между его рождением и смертью, существуют ограничения на разницу возраста человека и его родителей, человек не может служить в организации вне периода ее существования и т. п.

Опыт использования комплекса Фактограф в конкретных исследовательских проектах (например, [8]) продемонстрировал существенное повышение эффективности деятельности научного работника. Комплекс Фактограф предполагает принципиально открытое формирование модели предметной области. В существующих применениях комплекса Фактограф рассматриваются следующие объекты классов: лицо; организация; географический; документальный. В других применениях может быть рассмотрен другой набор классов, включающий технические, культурные или природные объекты, интересующих исследователя категорий. Соответственно классам объектов формируются наборы классов свойств и их возможных значений, а также накладываемых на них ограничений.

Приведем основные характеристики комплекса Фактограф на настоящее время. Реализованы: 43 интерфейсные формы; 12 отчетов (включая подчиненные формы и отчеты); 80 запросов на языке SQL; 6 модулей, объем которых вместе с кодом форм и отчетов составляет около 5000 строк кода на языке Visual Basic.

Масштабы коллекций π-фактов в существующих применениях комплекса Фактограф измеряются тысячами объектов и десятком тысяч их свойств, что вполне укладывается в ресурсные возможности используемой СУБД MS Access. В то же время сама модель π-фактов концептуально не реляционная. Отношения между объектами в ней введены посредством записей ключ-значение, а не введением новых столбцов и таблиц. Выбор MS Access был продиктован исключительно соображениями доступности. Для крупномасштабных применений потребуется переход на современные средства управления данными, например, на соответствующую концептуальной модели π-фактов парадигму NoSQL [9].

Заключение

В работе модель фактоподобных высказываний представлена на концептуальном уровне, а основанная на ней информационная технология описана в общих чертах. Возможные применения модели не ограничиваются рамками конкретного инструментального комплекса Фактограф. Модель может быть выражена в терминах другой среды, и для ее реализации применимы в том числе современные языки онтологий. Опора на языки онтологий позволит организовать непосредственный обмен знаниями с информационными системами семейства открытых связанных данных.

Список литературы

1. **Semantic Web**. URL: http://www.w3.org/standards/semanticweb
2. **The DBpedia** Knowledge Base. URL: http://dbpedia.org/About
3. **The Linking** Open Data cloud diagram http://lod-cloud.net/
4. **Маркова Н. А.** Формализация фактоподобных высказываний в конкретно-исторических исследованиях // Труды XVI конференции Электронные библиотеки: перспективные методы и технологии, электронные коллекции. Дубна: Изд-во ОИЯИ Дубна, 2014. С. 98—103. URL: http://ceur-ws.org/Vol-1297/098-103_paper-16.pdf
5. **Маркова Н. А**. Логика биографических фактов // Информатика и ее применения. 2012. Т. 6, Вып. 2. С. 49—58.
6. **Функциональные** требования к библиографическим записям: окончат. отчет / Рос. библ. ассоц., Рос. гос. б-ка; пер. с англ. В. В. Арефьев; науч. ред. Т. А. Бахтурина, Н. Н. Каспарова, Н. Ю. Кулыгина. М.: Пашков дом, 2008. 166 с.
7. **Маркова Н. А.** Программа Средства интеграции, хранения и анализа биографических данных (Фактограф). Свидетельство о государственной регистрации программы для ЭВМ № 2013617234 от 06.08.2013.
8. **Маркова Н. А.** База данных Приходское духовенство Коломенской округи XVIII- начала XIX веков. Свидетельство о регистрации базы данных № 2013620656 от 28.05.2013.
9. **Strauch C., Kriha W**. NoSQL databases. URL: http://www.christof-strauch.de/nosqldbs.pdf

N. A. Markova, Senior researcher, e-mail: MarkovaNatAlex@gmail.com,
Institute of Informatics Problems, FRC CSC RAS, Moscow,

Support Technology for Specific Historical Studies on the Base of Fact-like Propositions Model

Specific historical studies represent researches of biographies, local history, institution history, genealogy, background of scientific and cultural explorations.

The paper proposes a formal model of fact-like propositions that specify not only true statements, but suggestions, hypothesis, incomplete information, the results of analytic/synthetic processing of specific historical studies. The model fixes object characteristics both relational and literal. It takes into account temporal features and logical connectives. The model considers derivation chains from interpretation of sources through reasoning to representation of results in target documents.

The model is the base of supporting IT. Short description of the tool named "Factograph" is provided. The current applications of "Factograph" consider Person, Institution, Geographical entity and Document as object classes. Nomenclature of object classes, nomenclature of their characteristics as well as their possible values and restrictions imposed on them are open and can be changed at any moment.

Operations of "Factograph" have demonstrated high efficiency of the specific historical studies.

Keywords: *specific historical studies, object model, fact-like proposition, information technology*

References

1. **Semantic** Web, available at: http://www.w3.org/standards/semanticweb

2. **The DBpedia** Knowledge Base, available at: http://dbpedia.org/About

3. **The Linking** Open Data cloud diagram, available at: http://lod-cloud.net/

4. **Markova N. A**. Formalizatsiya faktopodobnykh vyskazyvanii v konkretno-istoricheskikh issledovaniyakh. *Trudy XVI konferentsii Elektronnye biblioteki: perspektivnye metody i tekhnologii, elektronnye kollektsii.* Dubna: Izd-vo OIYaI Dubna, 2014, pp. 98—103, available at: http://ceur-ws.org/Vol-1297/098-103_paper-16.pdf (in Russian).

5. **Markova N.A** Logika biograficheskikh faktov. *Informatika i ee primeneniya,* 2012, vol. 6, issue 2, pp. 49—58 (in Russian).

6. ***Funktsional'nye*** *trebovaniya k bibliograficheskim zapisyam: okonchat. otchet* (Functional requirements for bibliographic records / Rus. bibl. assoc.), per. s angl. V. V. Aref'ev; eds. T. A. Bakhturina, N. N. Kasparova, N. Yu. Kulygina. Moscow: Pashkov dom, 2008, 166 p. (in Russian).

7. **Markova N. A.** *Programma Sredstva integratsii, khraneniya i analiza biograficheskikh dannykh (Faktograf).* (Software tool for integration, storage and analysis of biographical data (Factograph)). The certificate of registration of software № 2013617234 dated 06.08.2013 (in Russian).

8. **Markova N. A.** *Baza dannykh Prikhodskoe dukhovenstvo Kolomenskoi okrugi XVIII- nachala XIX vekov.* (Data base: Parish clergy Kolomna districts XVIII—XIX centuries). The certificate of registration database № 2013620656 dated 28.05.2013 (in Russian).

9. **Strauch C., Kriha W.** NoSQL databases, available at: http://www.christof-strauch.de/nosqldbs.pdf

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4
Технический редактор *Е. М. Патрушева.* Корректор *М. Г. Джавадян*

Сдано в набор 05.03.2015 г. Подписано в печать 22.04.2014 г. Формат 60×88 1/8. Заказ PI515
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
119071, г. Москва, Ленинский пр-т, д.19, стр.1. Сайт: **www.aov.ru**