ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г.

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 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.

УДК 004.75

В. А. Васенин, д-р физ.-мат. наук, зав. лаб., **С. А. Афонин**, канд. физ.-мат. наук, вед. науч. сотр., НИИ механики МГУ им. М. В. Ломоносова, **Д. Д. Голомазов**, аспирант, МГУ им. М. В. Ломоносова,

e-mail: vasenin@msu.ru

Использование семантических технологий для обнаружения Грид-ресурсов

Поиск доступных для удаленного использования ресурсов составляет важный этап процесса построения плана выполнения распределенных на сетевой среде композитных приложений. Настоящая работа посвящена частному случаю решения этой задачи, а именно — обнаружению ресурсов в Грид-системах. В работе представлен краткий обзор существующих на этом направлении подходов. Предлагается метод обнаружения ресурсов, основанный на формальном семантическом описании прикладных задач, которые решаются с помощью Грид-сервисов. В работе представлены примеры использования предлагаемого подхода.

Ключевые слова: *онтология, семантический Грид, обнаружение ресурсов*

Введение

В процессе проектирования приложений системные архитекторы стоят перед вопросом выбора готовых для использования компонентов, решающих отдельные задачи разрабатываемого приложения. Наличие большого числа ресурсов в виде системных, общецелевых и прикладных сервисов, доступных как для удаленного использования, так и распространяемых в виде автономных программных модулей, комплексов и библиотек, значительно осложняет выбор тех из них, которые наиболее точно соответствуют потребностям создаваемой системы. В области Грид-вычислений выбор еще более усложняется, так как подобный поиск необходимо проводить в процессе выполнения приложения, а доступные сервисы динамически добавляются и удаляются из вычислительной среды, обслуживающей приложение. Задача обнаружения ресурсов (*Resource Discovery*) является важной и для работы Грид-систем в целом. Специалисты и организации, создающие сервисы, заинтересованы в том, чтобы пользователи находили и использовали эти ресурсы. Для этого им нужно оповестить потенциальных пользователей о существовании сервиса с определенными характеристиками и функциональными возможностями. В рамках Грид-системы данную задачу призвана решать служба обнаружения ресурсов, которая должна предоставлять пользователям следующую информацию:

- каталог, по которому пользователь может посмотреть существующие и доступные для него сервисы;
- информация о конкретном сервисе, а именно
  - интерфейс доступа к сервису, метод доступа, имена функций, параметров и возвращаемых значений;
  - технические характеристики ресурса, на котором работает сервис (например, производительность и ограничения на объем данных);
  - назначение сервиса, а именно функции, которые он может выполнять.

Рассмотрим потенциально возможный сценарий поиска ресурсов, необходимых для решения научных задач. Будем предполагать, что ученый, использующий некоторые сервисы, является специалистом в своей предметной области, однако не является экспертом в области информационных технологий. Его цель — построение программного комплекса, решающего его прикладную задачу, в максимальной степени используя для этого уже существующие сервисы и другие ресурсы. Возможная последовательность необходимых для этого действий может включать следующие этапы:

- составление общего плана решения прикладной задачи и ее декомпозиция на отдельные подзадачи;

- поиск существующих сервисов, доступных и решающих отдельные подзадачи или их части;
- разработка недостающих сервисов;
- компоновка программного комплекса.

Такой процесс может носить итерационный характер. В результате анализа доступных сервисов (который включает анализ как их алгоритмических и вычислительных возможностей, так и аппаратных ресурсов, например объема доступной оперативной памяти) может появиться необходимость в изменении первоначального плана решения прикладной задачи. Наибольшую сложность в этом процессе представляет вопрос поиска существующих сервисов. Для реализации эффективного поиска Грид-сервисов необходимо создание методов *описания* сервисов, *распространения* этой информации и реализации собственно *поиска* сервиса по запросу пользователя.

Одним из наиболее перспективных подходов к описанию и поиску Грид-ресурсов является подход *Semantic Grid* [6] в рамках которого для спецификации характеристик сервиса используются онтологии. Отметим, что работы, посвященные применению семантических технологий для решения рассматриваемой задачи, в основном нацелены на формальное описание первых двух из перечисленных выше типов характеристик сервиса, а именно интерфейса доступа к нему и технических параметров вычислительного узла, на котором он работает. Публикаций, посвященных семантическому описанию *назначения* сервиса, т. е. спецификации вычислительной задачи, которую он решает, авторам найти не удалось. Настоящая работа призвана заполнить этот пробел на направлении *Semantic Grid*.

Методы описания и поиска Грид-ресурсов

Службы поиска Грид-ресурсов можно разделить на два класса: распределенные службы, которые не имеют выделенного сервера для размещения информации о доступных ресурсах, и централизованные каталоги.

Большинство современных систем поиска ресурсов относятся к первому классу. Наиболее успешной попыткой построения централизованного каталога сервисов являлся коммерческий проект UDDI (*Universal Description, Discovery and Integration*) [4], основными разработчиками которого были компании Microsoft, IBM и SAP. После интенсивного развития в 2002—2005 гг. он утратил свою популярность, главным образом из-за требования централизации каталога описаний. В 2006 г. основные участники объявили о выходе из проекта. Доступные в Интернет каталоги веб-сервисов включают достаточно большое число служб. Например, каталог *seekda* содержит свыше 28 000 сервисов. Однако каждая запись включает очень ограниченную информацию, например, название организации-поставщика сервиса, названия методов и их параметров, полученные из WSDL-файлов, ключевые слова и краткое текстовое описание.

Значительное распространение получил подход "поисковых систем", предполагающий, что владельцы ресурсов составляют их описание, а некоторый поисковый агент (аналогичный роботам, используемым в поисковых системах, таких, как Google) находит и индексирует такие описания. В данном случае поисковый сервер выступает в роли централизованного каталога, однако информация вводится децентрализованно.

В проекте TeraGrid 11], поддерживаемом организацией Grid Infrastructure Group (Университет Чикаго, США), на основе опыта его исполнителей сформулированы следующие требования, которым должна удовлетворять информационная система обнаружения и поиска Грид-ресурсов:

- информация в системе должна быть не менее актуальна, чем документация, сопровождающая сервис;
- участники сети должны иметь контроль над информацией о своих ресурсах;
- система описания ресурсов должна допускать внесение данных, отражающих специфику ресурса;
- для выполнения некоторых операций необходима авторизация пользователей (например для просмотра очереди заданий своего сервиса);
- необходима поддержка широкого набора протоколов (WS/SOAP, WS/REST) и форматов (HTML, XML, CSV, JSON).

Рассмотрим две наиболее широко распространенные децентрализованные системы описания ресурсов. Первая из них *Monitoring and Discovery Service* (MDS) [5] — используется в инструментальном комплексе Globus Toolkit 4 [8] и является распределенной системой, предоставляющей технологию мониторинга и обнаружения ресурсов и сервисов. Вторая — *Open Cloud Computing Interface* (OCCI) [12] — представляет собой сетевой протокол для управления облачной инфраструктурой.

Система MDS состоит из индексов сервисов, организованных в иерархическую структуру [5]. Для включения в каталог создатель сервиса запускает индекс-сервис, который распространяет информацию о ресурсе и сервисе другим индекс-сервисам. Постепенно информация стекается в главный индекс-сервис. Заметим, что MDS использует модель, включающую и ресурсы, и сервисы (как атрибуты ресурсов). В качестве формата хранения и передачи данных используется XML. В случае, когда доступ к сервисам осуществляется по протоколу HTTP, в заголовках ответа может передаваться дополнительная информация о сервисе, в частности характеристики ресурса, на котором он работает. Основными достоинствами MDS являются четкая формальная модель и большой набор функциональных возможностей (например поиск сервисов). Главным недостатком MDS является сложность освоения и работы со службой. Отметим, что в разрабатываемом в настоящее время комплексе Globus Toolkit 5 предпочтение отдается новому средству *Integrated Information Service* (IIS). Одна из причин в том, что для спецификации характеристик веб-сер-

висов в Globus Toolkit 4 использовалась технология WSRF (*Web Services Resource Framework*), представляющая собой набор соглашений для описания веб-сервисов, а также технология WSRP (*Web Services for Remote Portlets*) — сетевой протокол взаимодействия веб-сервисов с портлетами (подключаемыми компонентами пользовательского интерфейса). Однако в версии Globus Toolkit 5 от использования WSRF и WSRP отказались.

OCCI — это сетевой протокол для управления облачной инфраструктурой, разрабатываемый *Open Grid Forum* (OGF). Данная технология является относительно новой. В отличие от MDS, протокол OCCI предоставляет только средство описания сервисов, не затрагивающее решение задач индексации и каталогизации, которые необходимо решать отдельно. Основная идея подхода заключается в описании сервиса в обычном HTML-файле, содержащем характеристики сервиса и ссылки на него. Метаинформация, например свойства и характеристики сервиса, добавляется в виде значений специальных атрибутов тегов $\langle a \rangle$, таких как *rel*. Такое описание, оставаясь вполне пригодным для чтения пользователем с помощью браузера, становится также доступным и для обработки ЭВМ. Для создателя файла добавление метаинформации является относительно легкой задачей, что выгодно отличает подход от MDS. Сочетание простоты и достаточно высоких выразительных возможностей является основным достоинством OCCI. Другим преимуществом OCCI является тот факт, что HTML-страницы с описанием сервисов могут ссылаться на другие веб-страницы (и наоборот), попадая тем самым в Интернет. Это обстоятельство позволяет применять для обнаружения и классификации сервисов стандартные методы тематического анализа текстов, анализа ссылок и подобные им. В настоящее время внедрение технологии OCCI находится на стадии, когда владельцы сервисов создают HTML-файлы, в которых описывают их и ссылаются на другие сервисы, однако поисковых служб пока нет.

Отметим, что ни одно из описанных выше решений не предназначено для формального задания назначения сервиса. И в MDS, и в OCCI существует техническая возможность задания произвольных значений атрибутов сервисов, однако нет никаких соглашений, кроме самых тривиальных, регулирующих методику классификации сервисов. По этой причине автоматический подбор сервисов на основе задач, которые они решают, при использовании рассмотренных систем в настоящее время в практическом смысле невозможен, поскольку один и тот же сервис может быть описан по-разному.

Семантический подход к решению задачи обнаружения и поиска Грид-ресурсов

Под *Semantic Grid* обычно понимают набор методов и алгоритмов, позволяющих проводить поиск ресурсов с использованием формального описания их характеристик. В настоящем разделе представлено общее описание семантического подхода к решению задачи обнаружения и поиска Грид-ресурсов, которое включает формальные основы семантических технологий, а также краткий обзор результатов исследований на этом направлении.

Формальные основы семантических технологий

Распространенным методом формального представления онтологий являются различные дескриптивные логики. Под онтологией в данном случае понимается пара $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, где $\mathcal{T}$ содержит универсальные утверждения, а $\mathcal{A}$ — утверждения об индивидуальных объектах. Дескриптивные логики, использующиеся для описания онтологий, отличаются друг от друга набором синтаксических конструкций, допустимых при формировании утверждений из множеств $\mathcal{T}$ и $\mathcal{A}$. Рассмотрим в качестве иллюстрации синтаксис логики $\mathcal{ALCOQ}b_{Reg}$ [2]. Пусть **C**, **R**, **I** — счетные множества, именуемые множествами *концептов, отношений* и *имен объектов соответственно*. *Атомарные концепты B*, *концепты C*, *простые отношения S* и *отношения T* подчиняются следующей грамматике, где $a \in \mathbf{I}$, $A \in \mathbf{C}$, $n \in \mathbb{N}$:

$$B ::= A \mid \{a\}$$

$$C ::= B \mid \neg C \mid C \sqcap C \mid C \sqcup C \mid \forall T.C \mid \exists T.C \mid$$

$$\leqslant nS.C \mid \geqslant nS.C$$

$$S ::= S \cap S \mid S \cup S \mid S \backslash S$$

$$T ::= S \mid T \cup T \mid T \circ T \mid T^* \mid id(C).$$

Множество $\mathcal{A}$ содержит *утверждения* вида $C(a)$, $S(a, b)$ и $a \neq b$, где $C \in \mathbf{C}$, $S \in \mathbf{R}$ и $a, b \in \mathbf{I}$ (принадлежность индивидуальных объектов заданным концептам и отношениям). Множество $\mathcal{T}$ является конечным множеством *аксиом вложения концептов* $C_1 \sqsubseteq C_2$ и *аксиом вложения отношений* $S_1 \sqsubseteq S_2$, где C_1 и C_2 — концепты, а S_1 и S_2 — простые отношения.

Под *интерпретацией* понимается пара $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, где $\Delta^{\mathcal{I}}$ — непустое множество, именуемое *доменом*, а $\cdot^{\mathcal{I}}$ — интерпретационная функция, сопоставляющая именам объектов элементы домена, концептам — подмножества домена, а отношениям — подмножества декартова произведения $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Выражения, определяющие концепты и отношения, интерпретируются естественным образом, например ($x, y, z \in \Delta^{\mathcal{I}}$):

$$(T_1 \circ T_2)^{\mathcal{I}} = \{(x, z) \mid \exists y (x, y) \in T_1^{\mathcal{I}} \wedge (y, z) \in T_2^{\mathcal{I}}\},$$

$$(id(C))^{\mathcal{I}} = \{(x, x) \mid x \in C^{\mathcal{I}}\},$$

$$(\exists T.C)^{\mathcal{I}} = \{x \mid \exists y\ (x, y) \in T^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\},$$

$$(\leqslant nS.C)^{\mathcal{I}} = \{x \mid |\{y \mid (x, y) \in S^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}| \leqslant n\}.$$

Отношение T^* интерпретируется как рефлексивное и транзитивное замыкание T (итерация Клини). Заметим, что отношения T определяют регулярные языки над алфавитом простых отношений S, и индекс Reg в названии логики $\mathcal{ALCOQ}b_{Reg}$ обозначает такую возможность.

Интерпретация $\mathcal{I}$ *удовлетворяет* аксиоме вложения концептов $C_1 \sqsubseteq C_2$ (или аксиоме вложения отношений $S_1 \sqsubseteq S_2$), если $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ (или $S_1^{\mathcal{I}} \subseteq S_2^{\mathcal{I}}$). Интерпретация $\mathcal{I}$ является *моделью* онтологии $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, если она удовлетворяет всем утверждениям из $\mathcal{A}$ и аксиомам из $\mathcal{T}$ онтологии $\mathcal{K}$. Отметим, что интерпретация онтологии может быть естественным образом представлена в виде ориентированного графа с помеченными ребрами.

Преимущество использования логического формализма при описании онтологий состоит в возможности проведения логического вывода: по заданной (частичной) информации о свойствах индивидуальных объектов (утверждений множества $\mathcal{A}$) и "универсальных" правил, заданных во множестве $\mathcal{T}$, можно автоматически получать новые утверждения об индивидуальных объектах. Пример такого вывода в области Грид будет приведен далее.

Применение онтологий к задаче обнаружения и поиска Грид-ресурсов

Задача обнаружения Грид-ресурсов может быть разделена на две практически независимые подзадачи, а именно — *распространение* информации о новых ресурсах в распределенной сети и *обработка* этой информации.

Отметим, что наиболее распространенные системы описания Грид-сервисов (рассмотренные выше) обладают техническими возможностями, позволяющими использовать семантическую информацию при описании сервисов. Описание сервиса в службе MDS, входящей в состав Globus Toolkit, может содержать свойство, значением которого является произвольный XML-документ. Включение метаинформации в атрибуты HTML-тегов в рамках технологии OCCI позволяет переводить ее в формат RDF и применять уже разработанные подходы и методики в области семантических технологий, например, алгоритмы хранения, интеграции RDF-троек, поиска по ним с помощью языка SPARQL, а также логического вывода новых утверждений.

В рамках подхода *Semantic Grid* онтологии используются для решения различных задач. Наиболее распространенным применением онтологий является задача поиска вычислительных ресурсов. Например, если пользователь выполняет поиск ресурса под управлением операционной системы Linux, а службе MDS известно, что на некотором ресурсе установлен дистрибутив CentOS, то онтология может использоваться для автоматического логического вывода информации о том, что на этом ресурсе установлена операционная система Linux.

В проекте UniGrids [17], поддерживаемом Европейским союзом, выделяются три уровня онтологий, использующихся при поиске ресурсов [14]:

- base concepts — основные понятия Грид (ресурс, память, процессор);
- domain independent ontologies — понятия, относящиеся к той или иной среде промежуточного уровня (Globus Toolkit, gLite [10], UniGrids);
- domain ontologies — понятия, характерные для конкретных реализаций Грид (TeraGrid, NorduGrid [7]).

В работе [13] предлагается метод добавления семантической информации к системе MDS в Globus Toolkit 4, получивший название S-MDS (*Semantic Monitoring and Discovery Service*). Решение основано на построении онтологии понятий Грид и отображении описаний сервисов в эту онтологию. В качестве примера рассматривается запрос на поиск UNIX-совместимого ресурса. Отмечается, что свойства ресурса (в стандартном описании MDS) не содержат информации о совместимости. В цитируемой работе предлагается:

- метод автоматического сопоставления элементов описания сервисов элементам онтологии;
- архитектура системы хранения и обновления семантических описаний ресурсов (которые могут быть большого объема);
- средства пользовательского интерфейса, обеспечивающие взаимодействие с конечным пользователем.

Основное отличие S-MDS от похожей технологии S-OGSA (*Semantic Open Grid Architecture*) [3] состоит в том, что S-MDS ориентирован на автоматическое построение семантических метаданных. Недостатком подхода является то обстоятельство, что технология S-MDS предназначена для использования с Globus Toolkit, и перенос ее на другие системы затруднен.

Работа [1] посвящена управлению ресурсами в Грид-среде, в частности, задачам объединения, мониторинга и поиска ресурсов. Авторы этой публикации предлагают решение для поиска ресурсов на основе формального описания их характеристик с помощью онтологии. В качестве онтологии используется *Grid Resource Ontology*, созданная ими на языке OWL в редакторе *Protégé*, а для поиска ресурсов применяется средство логического вывода *Algernon*. Программная реализация решения включает в себя семантический компонент для интеграции с Globus Toolkit (а именно — с классическим планировщиком GridWay [9]), позволяющий представлять метаданные Грид на языке онтологий. GridWay — это инструментарий для диспетчеризации Грид-ресурсов, который позволяет собирать информацию о них, осуществляет совмещение требований приложения и ресурсов, а также распределяет задачи по этим ресурсам.

Отметим, что существуют и другие инструментальные средства, позволяющие решать задачу совмещения требований приложения и ресурсов, например, MDS, Condor [15], Gridbus Broker [16] и аналогичные

им. Однако ни одно из них не поддерживает семантические технологии, а все алгоритмы используют поиск ресурсов по ключевым словам. Например, если для решения своей задачи исследователю нужна машина, работающая под управлением операционной системы Linux, он вводит запрос *Linux*. Служба поиска не найдет машины, работающие под управлением операционной системы Fedora, хотя они тоже подходят, так как Fedora является дистрибутивом Linux. Для осуществления подобного логического вывода (Fedora это Linux) и используются онтологии.

В работе [1] отмечается, что предлагаемая система позволяет искать ресурсы (или сервисы), которые, например, могут решать дифференциальные уравнения заданного вида. Возможности поиска в данном случае ограничены лишь используемой онтологией, которая может представлять собой формальное описание как вычислительных ресурсов (включая такие понятия, как процессор, память, операционная система), так и направления области знаний (например, дифференциальные уравнения). Два перечисленных варианта принципиально ничем не отличаются. Заметим, что авторы публикации [1] заостряют свое внимание на формальном описании вычислительных ресурсов, а смысловое описание упоминается лишь вскользь. Отметим, что в работе рассматриваются именно технологии планирования и диспетчеризации при решении задач, а не поиска сервисов. Как следствие — основное внимание уделяется техническим характеристикам ресурсов, а не задачам, которые решают Грид-сервисы. Информация о ресурсах записывается в свойства сервисов и загружается в MDS.

В работе [18] онтологические методы интеграции данных применяются для вычисления запросов вида "найти все ресурсы, которые входят в виртуальную организацию *X*, имеют более 100 свободных процессоров, и на которых может быть запущено приложение *Z*". Для решения такой задачи авторы строят онтологию Грид (включающую такие понятия, как ресурс, процессор, память и подобные им), заполняют ее реальными данными из MDS и выполняют SPARQL-запрос. Построенная онтология позволяет интегрировать информацию из различных служб. Приведенные в работе результаты показывают, что полнота и точность поиска оказываются значительно выше, чем у традиционных систем обнаружения Грид-ресурсов.

Как уже было отмечено выше, в публикациях в рамках подхода *Semantic Grid* предлагается использовать онтологии для формального задания характеристик Грид-сервисов. В большинстве работ рассматриваются только *вычислительные характеристики* ресурсов, на которых работают сервисы. В настоящей работе предлагается расширить этот подход в целях формального описания *назначения* сервисов, т. е. задач, которые они решают. В следующем разделе представлено описание предлагаемого подхода и примеры, демонстрирующие его практическую значимость.

Использование семантических технологий для описания назначения Грид-сервисов

В рамках концепции *Semantic Grid* для формального смыслового описания Грид-сервисов предлагается использовать семантические технологии, в частности, выражать назначение и характеристики сервисов на языке онтологий.

Следует отметить, что существуют две стороны процесса функционирования Грид-сервиса — техническая и содержательная. Техническая сторона имеет дело с такими параметрами, как частота используемого процессора, число ядер, операционная система, загруженность сервиса, а также спецификация интерфейса доступа к нему. Содержательная сторона отражает назначение службы в терминах задачи, которую она решает. Например, сервис может реализовывать какой-либо алгоритм и описание содержательной стороны процесса его функционирования состоит в определении этого алгоритма, указании параметров его работы и других свойств.

Как видно из представленного в предыдущем разделе краткого обзора основных работ по *Semantic Grid*, в большинстве таких работ в настоящее время рассматривается лишь техническая сторона сервисов, и для ее описания предлагается использовать онтологии. Семантические технологии действительно дают ряд преимуществ при задании метаданных, их распространении и при организации поиска по ним. Однако, по мнению авторов, более широкие возможности для развития *Semantic Grid* открываются именно на направлении семантического описания *содержательной* стороны функционирования Грид-сервисов. В качестве обоснования этого утверждения рассмотрим следующий сценарий использования Грид-системы.

Предположим, что исследователю, который занимается моделированием течения жидкости внутри объектов сложной формы (например трубах), необходимо численно решить дифференциальное уравнение в частных производных четвертого порядка вида $\Delta\Delta\varphi = 0$ в некоторой сложной области. Этот класс уравнений имеет специальное название *"бигармонические уравнения"*. Для решения этой задачи подходят далеко не все методы решения дифференциальных уравнений. Если исследователь не знает, какой метод ему наиболее подходит в данном случае, ему придется вручную просматривать каталог сервисов, посвященных решению дифференциальных уравнений (который может включать в себя сотни сервисов), читать описание каждой службы и оценивать, может ли она решить это уравнение. Отметим, что поиск по таким параметрам, как частота процессора и объем оперативной памяти, при наличии семантического описания *технической* стороны сервисов, может быть осуществлен сравнительно легко. Однако в данном случае его явно недостаточно. Исследователю нужен сервис, который позволяет решать его конкретную задачу, и для автоматизированного подбора такого сервиса необходимо семантическое описание *содержа-*

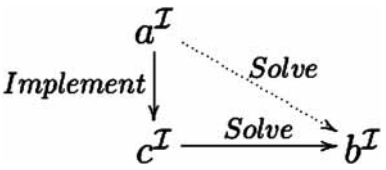
тельной стороны функционирования Грид-служб. Когда же концепция содержательного описания Грид-сервисов будет реализована, что включает в себя построение онтологий областей знаний, аннотирование сервисов с помощью этих онтологий и индексирование созданных метаданных поисковыми системами, тогда подбор нужного сервиса будет сводиться к формальной спецификации задачи путем ее автоматизированного выбора из онтологии и задания параметров. В рассматриваемом примере пользователь вводит запрос *"бигармоническое уравнение"*. Система находит это понятие в имеющейся модели области знаний, выраженной в онтологии, и, при необходимости, уточняет его у пользователя (например, предлагая такие классы, как линейные уравнения, уравнения в частных производных, обыкновенные дифференциальные уравнения).

Пользователь выбирает класс *"уравнения в частных производных"*, и система автоматически находит наиболее важные характеристики выбранного класса, уравнений (например, гладкость решений, сложность области, однородность) и запрашивает у пользователя значения этих параметров. После этого она ищет в каталоге сервисы, позволяющие решать уравнения заданного класса с такими характеристиками. Важно отметить, что использование онтологий, в отличие от подходов, основанных на поиске по ключевым словам, позволяет находить объекты (в данном случае — сервисы), в описании которых напрямую не заданы слова поискового запроса. Для понимания того, что сервис способен решать поставленную задачу, необходимо сделать логический переход. Приведем в качестве примера сервис, позволяющий решать дифференциальные уравнения в частных производных методом конечных элементов. Этот метод эффективен для решения упомянутой выше задачи численного решения дифференциального уравнения в частных производных в сложной области. В описании сервиса, однако, может быть указано лишь то, что он реализует метод конечных элементов для решения уравнений в частных производных. В этом случае поиск по ключевым словам (например, по таким запросам, как *"бигармоническое уравнение"*, *"дифференциальные уравнения четвертого порядка"*, *"течение жидкости в трубе"* и аналогичным им) не позволит найти необходимый сервис. При использовании же семантических технологий поисковая система "знает", опираясь на формализованное описание предметной области с помощью онтологии (предметной областью в данном случае могут быть *"численные методы"*), что метод конечных элементов позволяет численно решать дифференциальные уравнения в частных производных высокого порядка на сложных областях. На основе этого знания она автоматически делает вывод, что найденный сервис способен решить бигармоническое уравнение четвертого порядка на области, заданной пользователем, и выводит его в результатах поиска, эффективно решая задачу автоматизированного подбора Грид-сервиса.

Приведем формальное описание рассмотренного выше примера на языке дескриптивной логики. Пусть задана онтология $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, которая содержит концепты *Method* — "метод решения задач", *Problem* — "задача", *GridServise* — "Грид-сервис" и отношения *Solve* и *Implement*, где *Solve*(*a*, *b*) означает, что *a* решает задачу *b*, а *Implement*(*a*, *b*) означает, что *a* реализует метод *b*. Пусть множество $\mathcal{T}$ содержит утверждение $Implement \circ Solve \sqsubseteq Solve$, означающее, что сервис, реализующий метод, который решает некоторую задачу, сам решает эту задачу. Пусть множество $\mathcal{A}$ содержит следующие утверждения:

$$GridService(a),\ Problem(b),\ Method(c),\ Solve(c, b),\ Implement(a, c),$$

где *a* соответствует Грид-сервису, который реализует метод конечных элементов, *b* — задаче решения бигармонического уравнения, а *c* — методу конечных элементов. Пусть $\mathcal{I}$ — интерпретация, являющаяся моделью онтологии $\mathcal{K}$. Рассмотрим ориентированный граф интерпретации $\mathcal{I}$, вершины которого соответствуют интерпретациям экземпляров онтологии, а ребра — отношениям между ними (см. рисунок):



Сплошные ребра соответствуют отношениям, полученным на основании утверждений из множества $\mathcal{A}$. Пунктирное ребро было достроено на основании утверждения из $\mathcal{T}$, из которого следует, что в множестве $Solve^{\mathcal{I}}$ должна содержаться пара $(a^{\mathcal{I}}, b^{\mathcal{I}})$, так как интерпретация $\mathcal{I}$ является моделью онтологии $\mathcal{K}$.

Из представленного выше формального описания следует, что по запросу пользователя *"найти все сервисы, решающие бигармонические уравнения"* будет найден сервис *a*, хотя в его описании отсутствуют слова *"бигармонические уравнения"*, а лишь написано, что сервис реализует метод конечных элементов. С помощью знания, заключенного в онтологии (*"метод конечных элементов решает бигармонические уравнения"*), этот сервис будет выдан в качестве результата поиска. Отметим, что поиск по ключевым словам не позволит добиться требуемого результата, если только пользователь не введет запрос вида *"метод конечных элементов"*. Важно также подчеркнуть, что онтология выполняет роль эксперта, "консультирующего" пользователя по теоретическим аспектам предметной области, который может не знать, например, какими методами можно решать его задачу. В рассмотренном примере показано, какие формальные основы имеет под собой логический вывод новых утверждений, позволяющий выполнять сложные аналитические запросы без необходимости ручного задания всех связей

между объектами онтологии. Последнее обстоятельство выгодно отличает семантические технологии от традиционных инструментальных средств поиска по ключевым словам.

Для выполнения семантического поиска ресурсов необходимо создание онтологии области знания, включающей основные понятия, задачи и методы их решения. Рассмотрим возможную структуру онтологии в области численных методов. Отметим, что численные методы решения физических и механических задач составляют значительную часть Грид-приложений. Такая онтология должна содержать описания численных методов решения основных задач алгебры, математического анализа, дифференциальных уравнений и уравнений в частных производных. Например, конечный пользователь может выполнить поиск сервисов, реализующих алгоритмы численного решения уравнения теплопроводности. Разработчики сервисов могут опубликовать описания своих ресурсов, указав конкретные методы решения уравнений, например, схему Франкела—Дюфорта или схему Алена—Чена. Онтология позволяет автоматически определить, что уравнение теплопроводности является дифференциальным уравнением в частных производных параболического типа, а указанные численные методы могут использоваться для решения уравнений такого типа. После этого сервисы, реализующие эти методы, будут показаны пользователю. Для реализации такой схемы необходимо иметь информацию об основных задачах в общей постановке в рамках рассматриваемой области знания, их частных случаях и методах решения как общих, так и частных задач.

Заключение

В настоящей работе предлагается использовать семантические технологии для описания *назначения* Грид-сервисов. Эта идея является расширением подхода *Semantic Grid*, в рамках которого онтологии применяются для формального задания характеристик Грид-сервисов. В работе представлены примеры, демонстрирующие практическую значимость предлагаемого подхода. Отметим, что с технической точки зрения идея описания назначения Грид-сервисов не имеет препятствий, затрудняющих ее реализацию, так как широко распространенные службы поиска ресурсов (MDS и OCCI) позволяют задавать произвольные значения атрибутов сервисов. Для полноценной реализации предлагаемой концепции необходимо выполнить следующие шаги:

- создать или выбрать существующую онтологию рассматриваемой области знания;
- описать на ее языке сервисы в Грид-системе;
- создать инструмент поиска ресурсов (например, в виде дополнительного модуля к MDS), использующий семантическую информацию при поиске.

Отметим, что для решения последней задачи (создания инструмента поиска ресурсов) можно применить существующие технологии, предназначенные для работы с семантическими данными, например язык запросов SPARQL, а также программные средства, созданные на основе этих технологий.

Работа выполнена при частичной поддержке грантом РФФИ № 09-07-00366-а.

Список литературы

1. **Amarnath B. R., Somasundaram T. S., Ellappan M., and Buyya R.** Ontology-based Grid resource management // Software: Practice and Experience. 2009. № 39 (17). P. 1419—1438.

2. **Calyanese D., De Giacomo G., Lenzerini M., and Rosati R.** View-based Query Answering over Description Logic Ontologies // Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning. 2008. P. 242—251.

3. **Corcho O.** и др. An Overview of S-OGSA: A Reference Semantic Grid Architecture // Journal of Web Semantics. 2006. № 4. P. 102—115.

4. **Curbera F., Duftler M., Khalaf R., Nagy W., Mukhi N., and Weerawarana S.** Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI // Internet Computing, IEEE. 2002. № 6 (2). P. 86—93.

5. **Czajkowski K., Kesselman C., Fitzgerald S., and Foster I.** Grid information services for distributed resource sharing // Proc. of 10th IEEE International Symposium on High Performance Distributed Computing, IEEE Computer Society, 2001. P. 181—194.

6. **De Roure D., Jennings N. R., and Shadbolt N. R.** The semantic grid: Past, present, and future // Proc. of the IEEE. 2005. № 93 (3). P. 669—681.

7. **Ellert M., Konstantinov A., Kónya B., Smirnova O., and Wäänänen A.** The NorduGrid project: Using Globus toolkit for building Grid infrastructure // Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment. 2003. № 502 (2—3). P. 407—410.

8. **Foster I.** Globus toolkit version 4: Software for service-oriented systems // Proc. of IFIP International Conference on Network and Parallel Computing. 2005. P. 2—13.

9. **Huedo E., Montero R. S., and Llorente I. M.** The Grid Way framework for adaptive scheduling and execution on grids // Scalable Computing: Practice and Experience. 2001. № 6 (3). P. 1—8.

10. **Laure E., Fisher S., Frohner A.** et al. Programming the Grid with gLite // Computational Methods in Science and Technology. 2006. № 12 (1). P. 33—45.

11. **Liming L., Navarro J.-P., Blau E.** et al. TeraGrid's integrated information service // Proc. of the 5th Grid Computing Environments Workshop, ACM. 2009. P. 1—10.

12. **Metsch T., Edmonds A., and Nyrén R.** Open Cloud Computing Interface — Core. April 2011. URL: http://ogf.org/documents/GFD.183.pdf.

13. **Pahlevi S. M., and Kojima I.** S-MDS: Semantic Monitoring and Discovery System for the Grid // J. Grid Computing. 2009. № 7 (2). P. 205—224.

14. **Parkin M., van den Burghe S., Corcho O., Snelling D., and Brooke J.** The knowledge of the grid: A grid ontology // Proc. of the 6th Cracow Grid Workshop. 2006. P. 111—118.

15. **Thain D., Tannenbaum T., and Livny M.** Distributed computing in practice: The Condor experience // Concurrency and Computation: Practice and Experience. 2005. № 17 (2—4). P. 323—356.

16. **Venugopal S., Buyya R., and Winton L.** A grid service broker for scheduling distributed data-oriented applications on global grids // Proc. of the 2nd workshop on Middleware for grid computing, ACM. 2004. P. 75—80.

17. **Shah K.** UniGrids — Uniform Interface to Grid Services // Focus. 2004. № 13. P. 20—22.

18. **Xing W., Corcho O., Goble C., and Dikaiakos M. D.** An ActOn-based semantic information service for Grids // Future Generation Computer Systems. 2010. № 26 (3). P. 324—336.

УДК 004.05

С. А. Полицын, аспирант, "МАТИ" — РГТУ имени К. Э. Циолковского,
e-mail: pul_forever@mail.ru

Подходы к вычислению временных затрат на проекты в сфере разработки программного обеспечения на основе использования прецедентов

Оценка необходимого для реализации проектов времени, ресурсов и бюджета — важная задача для коллективов разработчиков. Обычно она решается приближенно с большой погрешностью. Одним из возможных путей решения может быть построение полученной на основе анализа данных о ранее выполненных проектах экспериментальной зависимости числа обнаруженных дефектов от времени. Предлагаются методика прогнозирования хода процесса разработки, а также подход, позволяющий с достаточной точностью проводить вычисление момента окончания проекта.

Ключевые слова: *управление проектами, расчет временных затрат, срок окончания проекта*

Анализ подходов к разработке и определению временных рамок проектов разработки программного обеспечения

Вопрос оптимальной организации разработки программного обеспечения (ПО) на сегодняшний день полностью не решен. Уже несколько десятилетий подходы сменяют друг друга с большой, особенно для инженерного дела, частотой: на смену водопадной модели (и ее модификациям) быстро пришли спиральная и итерационная модели, последние годы набирают силу "гибкие" модели, ориентированные на изменяющиеся требования заказчика и окружающей среды (*Agile*, *XP*, *SRUM*). Едва ли не каждый следующий выполненный проект отличается от предыдущего, даже если они оба предназначены для одного сегмента рынка и имеют схожие технические задания (ТЗ)[1]. Это тем более верно, если речь идет о новом для той или иной команды направлении деятельности.

В начале каждого проекта должны быть решены задачи определения его ресурсоемкости, времени реализации, бюджета и, соответственно, возможности выполнения при выбранном сочетании этих параметров. Эти задачи решаются приближенно, поскольку существующие методики расчета предлагают только грубые оценки, а необходимые ресурсы определяются исключительно на основе опыта и субъективного мнения людей, выступающих в роли экспертов[2], далеко не всегда являющихся специалистами в данной области.

Исходя из результатов переговоров как внутри команды, так и с заказчиком, который зачастую сам еще не до конца представляет требуемые характеристики будущей системы или попросту не знаком с современными достижениями в области информационных технологий, выясняются основные детали проекта, на основе которых определяются основные характеристики проекта.

[1] В данном случае под ТЗ понимается любое формальное или не формальное описание функций, исполняемых создаваемой программой, это может быть либо ТЗ в понимании ГОСТа, либо Use-Case модель Rational Unified Process, либо "протоколы совещаний команды", если речь идет об Agile-методике.

[2] Обычно речь идет о совете экспертов, состоящем из аналитика, бизнес-аналитика, ведущего разработчика, менеджера проекта, менеджера по тестированию.

В первую очередь, это "рамки" проекта (*Scope*) — совокупность задач по проекту с учетом набора свойств, которым должно обладать разрабатываемое приложение. Они определяют, что должно быть сделано, т. е. если, например, стоит задача разработки программного обеспечения банкомата, то необходимо определить, какими функциями оно должно обладать ("посмотреть счет", "снять деньги", "совершить платеж" и т. д.). Трудозатраты и ресурсы, необходимые для реализации этих функций, в дальнейшем и будут оцениваться при планировании проекта. Также обязательно выработать процедуру изменения (почти всегда расширения) рамок проекта, поскольку сколь бы качественно и досконально не было проведено начальное планирование, новые пожелания заказчика, которые обязательно надо учесть в разрабатываемой системе, могут привести к значительному увеличению всех параметров разработки. Например, небольшая с виду проверка данных пользователя по удаленной базе клиентов предприятия как минимум требует учета нескольких исключительных ситуаций в дополнение к учету задержки работы с этой базой.

На основе каждого требования определяется ряд соответствующих ему параметров: приоритет, сложность реализации, новизна и др. В литературе [1, 2] приводятся различные параметры, по которым рассчитываются характеристики проекта, а также методики определения их влияния на сроки его окончания, но, во-первых, все эти методики разрабатываются для конкретных проектов, во-вторых, на практике практически не используются из-за сложности расчетов и их низкой точности. Общим недостатком сложных методик является их базирование на упомянутом выше мнении экспертов. В практически каждом проекте выделяется два основных этапа: реализация функционала и отладка. Работы, выполняемые на первом этапе, хорошо структурированы (реализация функционала *A*, реализация подзадачи *A*1 и т. д.). Поэтому для предварительных расчетов на этом этапе предлагается использовать простейший вариант — линейную оценку времени, необходимого для реализации проекта. Для примера предлагается ограничить модель оценки тремя параметрами: сложностью выполнения, важностью точности вычислений и новизной. Тогда время разработки T_{dev} можно определить формулой

$$T_{dev} = K_{res}\sum_{n}(K_{diff}D + K_{pr}P_r + K_{inv}C_r + ...),$$

где K_{res} — коэффициент резервного времени, связан с учетом различных рисков проекта, среднего времени задержки выполнения задач в команде. В идеальном случае он может быть принят за 1, но на практике изменяется в пределах от 1,3 до 1,5; K_{diff} — коэффициент сложности задачи; D — сложность задачи, выраженная в человеко-часах; K_{pr}, P_r — коэффициент и параметр важности точности вычислений. В любых программах, в частности, ориентированных на финансовые операции, чрезвычайную важность приобретает точность вычислений. Известен случай, когда из-за возможной ошибки в одном из младших разрядов при вычислениях с плавающей точкой компания *Intel* проводила кампанию по отзыву своих процессоров; K_{inv}, C_r — коэффициент и параметр новизны решения; n — число требований к системе.

Число факторов и степень их влияния на проект не являются жестко заданными, однако для конкретного проекта могут быть выведены аналитически, например, с использованием корреляционных зависимостей: время добавления в проект дополнительных идентичных SQL-запросов линейно зависит от числа запросов, а необходимость изучения ранее не применявшейся технологии в ходе выполнения проекта (например, сторонним разработчиком была предоставлена новая библиотека функций) может привести к экспоненциальному росту времени и даже явиться причиной невозможности выполнения поставленной задачи.

Анализ планов проектов и определение недостатков планирования

Ни в одном из исследованных проектов на этапе отладки и тестирования не были выделены подэтапы, а также приблизительно определено необходимое для исправления ошибок время. Иногда вообще отладка выделяется как отдельный "черный" ящик уже по окончании реализации всего функционала. Понятно, что такие проекты будут ощущать нехватку ресурсов.

Независимо от выбранной методологии, после оценки времени, требуемого для разработки функционала программы, необходимо оценить время, которое будет затрачено на тестирование и исправление дефектов. В случае использования автоматизированных методов тестирования время, необходимое для написания тестовых сценариев, можно оценить аналогично времени разработки. Поскольку редкий продукт выпускается для работы исключительно на одинаковом оборудовании и при одинаковых настройках операционной системы, помимо времени выполнения тестовых сценариев необходимо также учитывать затраты на конфигурационное тестирование.

Затраты на тестирование в отдельной итерации проекта предлагается оценивать по следующей формуле:

$$T_t = N_{config}(\Sigma T_{case}) + K_{failed}N_{failed} + T_{avt},$$

где N_{config} — число проверяемых конфигураций; T_{case} — время, необходимое для выполнения одного сценария; K_{failed} — коэффициент, выражающий затраты

времени на исправление одной типичной ошибки; N_{failed} — число сценариев, пройденных с замечаниями; T_{avt} — затраты времени на автоматизированное тестирование (разработка, выполнение, анализ результатов, поддержка автоматизированных тестов).

Описанная выше методика оценки (или ее модификация) применяется в большинстве проектов. В результате удается спланировать распределение ресурсов и срок, необходимый для реализации требований, имеющихся на начало выполнения проекта. Однако на практике при рассмотрении проектов целиком эта методика не показала достаточно удовлетворительных результатов, в первую очередь, из-за очень грубых приближений при определении соответствующих коэффициентов. Кроме того, даже в небольших проектах разработка не заканчивается на реализации, за которой следует зачастую гораздо более длительная стадия модификаций требуемого функционала и исправления ошибок. Эту стадию сложно учесть в рамках вышеописанного подхода.

Необходимо разработать методы, которые позволили бы решить указанные проблемы за счет изменения и формализации процесса планирования, а также повысить его точность и прозрачность. Предлагаемые математические методы планирования проекта можно разделить на две независимые группы:

- математические методы, позволяющие повысить точность планирования реализации как функционала, так и автоматических тестов;
- методы прогнозирования времени окончания этапа исправления ошибок.

Методы второй группы используют данные о числе найденных в программном обеспечении ошибок, поэтому для успешного применения методов второй группы необходимо определить понятие ошибки в ПО, а также провести отбор ошибок, обязательных для исправления.

Разработка метода классификации ошибок для уточнения временных рамок проекта

Наиболее общее определение ошибки, являясь при этом сильно размытым, приведено в стандарте ISO: "ошибка — это несоответствие между целями пользователя и ответом системы" [3]. Ошибкой можно назвать недокументированные или нежелательные, "побочные" реакции программы на те или иные действия пользователя, а также при использовании ее одновременно с другими программами или на другой аппаратной платформе. Г. Майерс дает такое нестрогое определение: "Если программа не делает того, чего пользователь от нее вполне обоснованно ожидает, значит налицо программная ошибка" [4].

Высказывается мнение [5], что наиболее часто используемое определение ошибки как несоответствия между программой и ее спецификацией также не до конца верно. Программа, которая полностью соответствует своей спецификации, все равно может содержать ошибки в тех случаях, когда либо неверна спецификация, либо она не полностью отражает все детали программы. В целом многие авторы, например, приходят к наиболее общему нестрогому определению ошибки в программном обеспечении: "Дефектом может быть нечто, что не соответствует ожиданиям заказчика и что может быть, а может и не быть определено в спецификации программного продукта" [2].

Часто критикуется строгая спецификация [5], так как даже при ее наличии выявленные на этапе системных испытаний дефекты говорят о низком качестве программы. Исправление недостатков программы, особенно на заключительных этапах разработки, обязательно будет приводить к снижению надежности. Очевидно, что для разработки надежного программного обеспечения такой подход недопустим, однако проблемы наличия ошибок в спецификациях, субъективного оценивания пользователем качества программы существуют и не могут быть проигнорированы. Для программного обеспечения все проблемы, связанные с субъективной оценкой его качества, практически неизбежны.

Чтобы уменьшить влияние субъективности, следует разработать, во-первых, методику отбора дефектов системы, и, во-вторых, критерии качества программного обеспечения. Обе эти задачи часто являются источником разногласий внутри коллективов разработчиков. В статье приводится методика отбора дефектов, на основе которой можно делать выводы о готовности программного продукта к выпуску или завершению очередной итерации разработки.

В первую очередь, дефекты классифицируют по месту их возникновения [5]. Большая их часть — это именно функциональные ошибки, которые могут быть выявлены в ходе функционального тестирования программного продукта или сопутствующих артефактов (документации, спецификации, протоколы испытаний и т. п.). Вторым признаком, по которому может быть проведена классификация ошибок, является ранжирование с точки зрения пользователя. При таком способе можно выделить: ошибки проектирования; предложения по улучшению программы; расхождение с документацией; вопросы взаимодействия с аппаратурой; спорное поведение программы. К сожалению, в общем случае нет убедительных оснований относить ошибку к тому или иному разделу внутри такой классификации, так как обычно проблема носит комплексный характер. Стоит учитывать также вероятность ошибки классификации. Следствием подобных ошибок будет увеличение времени отладки программы. Для предотвращения этих ситуаций могут применяться системы автоматизированной классификации ошибок, функционирование которых должно опираться на использование методов автоматиче-

Расстановка приоритетов ошибки

Классификация ошибок по влиянию на ПО	Нормальные условия	"Сверх-нормальные"	Специальные условия
Ошибки, вызывающие сбои в программе или операционной системе	1	1	2
Ошибки, приводящие к нарушениям в основных функциях системы	1	2	3
Ошибки, приводящие к нарушениям в дополнительных функциях системы	2	3	3

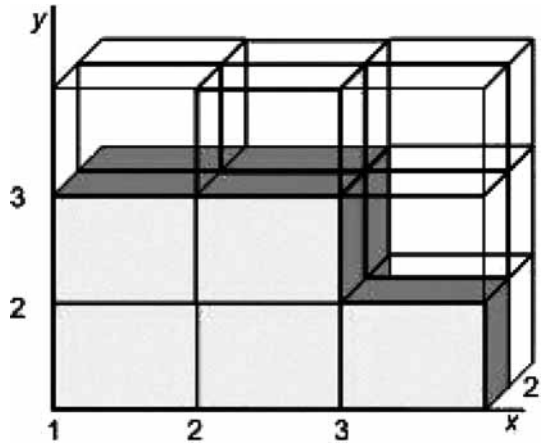


Рис. 1. Пример отбора дефектов на основе трех классификаций

ского анализа, в частности методов автоматической классификации текста. Другой вариант — классификация ошибок по степени их критичности, как правило, используется для программ, к которым предъявляются высокие требования по безопасности (или же для систем в целом).

Анализируя многообразие классификаций ошибок, можно сделать следующий вывод: ни одна из них в чистом виде не может быть применена, поскольку не отражает в полной мере действительно важные для проекта ошибки, требующие обязательного исправления. На этот вывод также наталкивает то, что не существует однозначного определения "действительно важной ошибки", вследствие чего возникает множество терминов. К числу таких терминов относятся "приоритет" (*priority*), "важность" (*importance*), "критичность", "серьезность" (*severity*), "значимость" (*significance*) и др. Разные авторы вкладывают в них различный смысл, но в действительности все они обозначают ошибку, исправление которой действительно необходимо.

На практике приходится работать одновременно с несколькими классификациями ошибок, выделяя важные ошибки по некоторому признаку из каждой. Это затрудняет процесс отбора ошибок. В то же время наличие разнообразных критериев, по которым оценивают ошибки, можно считать вполне допустимым, поскольку одну и ту же ошибку можно рассматривать с разных точек зрения.

По шкале каждого из критериев все имеющиеся ошибки ранжируются (таблица).

Такая схема является первым шагом к введению прозрачной шкалы для определения ошибки, обязательной для исправления. В общем случае требуется иметь одну функцию $f(x_1, x_2, ..., x_n)$, позволяющую быстро и наглядно определять необходимые к исправлению дефекты.

Это можно сделать на основе N-мерного массива, для каждой грани которого определена зависимость $f(x_n, x_{n+1})$. В простейшем случае может быть использовано произведение x_1 и x_2, или же, в зависимости от специфики проекта (см. таблицу) задана некоторая функция, которая определена для каждой классификации (рис. 1).

Аналогично можно поступить с другими параметрами, характеризующими конкретную ошибку. Таким образом, создается адаптированная к проекту объединенная классификация, которая позволяет быстро и прозрачно отбирать поступающие дефекты.

Методика расчета и уточнения данных о времени окончания итерации

Одним из путей решения проблемы планирования времени, необходимого для тестирования и отладки, может стать подход, основанный на использовании экспериментально полученной функции зависимости текущего числа активных[3] дефектов приложения от времени (рис. 2), полученной на основе данных о выполненных проектах. Для различных типов программных проектов, а также размера систем общий вид этой кривой и ее характеристики (наличие экстремумов, возрастание/убывание) является схожим.

На графике функции выделяется главный максимум, который достигается в окрестности момента времени T_0, когда выполнены 100 % требований спецификации. В этот момент число дефектов максимально. Это значение может быть либо оценено экспертным путем, либо

[3] Под активным понимается дефект на любом этапе своего жизненного цикла: "найден", "назначен", "исправлен", "на обсуждении", "на утверждении", за исключением статуса "закрыт".

Рис. 2. Распределение активных дефектов по времени

получено аналитически на основе данных о сложности задачи. Это относится и к локальным максимумам, которые соответствуют реализации не оговоренного заранее дополнительного функционала, или общедоступным версиям, которые вскрывают дополнительные дефекты за счет тестирования продукта широким кругом пользователей, а значит, для различных, зачастую непредвиденных или мало протестированных конфигураций или нового оборудования.

Эта кривая на бесконечности стремится к нулю. Целесообразно заранее определить порог, при достижении которого продукт считается завершенным. Он может быть определен, например, на основе подхода "Six Sigma" [6].

Таким образом, точка T_1, в которой значение функции будет равно некоторому определенному уровню, будет являться расчетным временем окончания проекта. Дальнейшая разработка при отсутствии новых требований или расширения рамок проекта является экономически нецелесообразной.

Определение и периодическое уточнение абсциссы точки T_1 предлагается проводить на основе методов аппроксимации кривых. Периодическая корректировка необходима ввиду воздействия на ход проекта внешних и внутренних факторов, таких как изменения в требованиях и новые конфигурации окружения, или недочеты в работе проектной команды на предыдущих этапах разработки. Более того, для предсказания времени гораздо важнее текущие (мгновенные) значения функции, чем более ранние данные. В ходе анализа выбрано оптимальное число предшествующих значений, на основе которых вычисляется текущая аппроксимирующая функция. Это значение равно, в зависимости от проекта, 3 или 4, разрабатываемое программное средство имеет настраиваемую опцию для этой величины. Таким образом, каждому моменту времени $T > 4$ можно поставить в соответствие число активных ошибок проекта N:

$T_{n-3}.....N_{n-3}$,

$T_{n-2}.....N_{n-2}$,

$T_{n-1}.....N_{n-1}$,

$T.....N$.

Анализ экспериментальных данных позволяет сделать предположение об аналитическом виде функции. В качестве приближающих функций можно предложить линейную зависимость при грубом приближении и дробно-рациональную функцию $y = x/(ax + b)$.

Первую из них можно использовать ввиду простоты и наглядности расчетов, а вторую исходя из требования:

$$\lim_{x \to \infty} y = 0.$$

В первом случае функция строится по методу наименьших квадратов ($y = kx + b$) и коэффициенты равны, соответственно:

$$k = \frac{n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i \right)^2},$$

$$b = \frac{1}{n} \left(\sum_{i=1}^{n} y_i - k \sum_{i=1}^{n} x_i \right).$$

Во втором случае приближающая функция имеет вид

$$y = \frac{x}{ax + b}. \qquad (1)$$

Ее можно преобразовать к виду

$$\frac{1}{y} = a + \frac{b}{a}, \; x > 0, \quad y > 0,$$

и если в исходном выражении заменить значения x и y обратными величинами

$$t = \frac{1}{x}, \quad q = \frac{1}{y},$$

то можно искать для новой функции приближающую функцию в виде линейной $q(t) = a + bt$. В этом случае задача сводится к предыдущей, и найденные значения a и b будут искомыми для формулы (1).

Возможность уточнения функции в каждый момент времени T_n делает модель адаптивной, позволяя сразу увидеть необходимость переноса сроков, перераспределения ресурсов и внесения других изменений, а также посмотреть последствия этих изменений.

Таким образом, предложенный подход позволяет, с одной стороны, прогнозировать сроки завершения, проекта целиком и динамически реагировать на происходящие изменения, с другой стороны, может быть применен в ходе планирования и для нахождения оптимального распределения ресурсов.

Список литературы

1. **Богданов В.** Управление проектами в Microsoft Project. СПб.: Питер, 2007. 592 с.

2. **Немировский И. Б., Старожукова И. А.** От стратегии до бюджета. М.: Диалектика, 2008. 510 с.

3. **Стандарт** ISO 9241-13.

4. **Майерс Г.** Искусство тестирования программ. М.: Финансы и статистика, 1982. 174 с.

5. **Канер С., Фолк Дж., Нгуен Е. К.** Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений: пер. с англ. М.: ДиаСофт, 2001. 544 с.

6. **Gygi C., DeCarlo N.** Six Sigma. John Wiley & Sons, Inc., 2005. 309 p.

ИНФОРМАЦИЯ

14—17 сентября 2012 в г. Харьков, Украина

10th IEEE EAST-WEST DESIGN & TEST SYMPOSIUM (EWDTS 2012)

Цель симпозиума **IEEE East-West Design & Test Symposium** (EWDTS) — расширение международного сотрудничества и обмен опытом между ведущими учеными Западной и Восточной Европы, Северной Америки и других стран в области автоматизации проектирования, тестирования и верификации электронных компонентов и систем.

Местом проведения EWDTS'12 является Харьковский национальный университет радиоэлектроники.

Важные даты: **Срок подачи докладов:** 15 июня, 2012
Итоги рецензирования: 1 августа, 2012

Регистрация докладов: **http://www.ewdtest.com/conf**

Адрес оргкомитета: Проф. Владимир Хаханов, кафедра Автоматизации проектирования вычислительной техники Харьковского национального университета радиоэлектроники, пр. Ленина 14, Харьков, 61166, Украина.

Тел.: +380-57-702-13-26, E-mail: hahanov@kture.kharkov.ua

УДК 004.451.2

Н. В. Заборовский, аспирант, **А. Г. Тормасов**, д-р физ.-мат. наук, проф.,
Московский физико-технический институт (государственный университет),
e-mail: turnik@mail.ru

Расчетный подход к статическому анализу программного кода на предмет наличия состояний гонки

Существуют различные методики анализа корректности многопоточных алгоритмов. Предложен и обоснован новый подход к проблеме поиска состояний гонки, основанный на статическом анализе программного кода и анализе вспомогательных конструкций, включающих небольшую вычислительную часть.

Ключевые слова: *статический анализ, состояние гонки, многопоточные алгоритмы*

Введение

Состояние гонки (*race condition*) — ситуация, когда несколько потоков одновременно обращаются к одному и тому же ресурсу, причем хотя бы один из потоков выполняет операцию записи, и порядок этих обращений точно не определен. Обнаружить наличие состояния гонки в коде очень непросто, так как о них можно и вовсе не знать, пока они не проявят себя. Процедура обнаружения состояния гонки может быть очень трудоемкой, даже если в наличии есть состояния всех потоков исполнения на момент некорректной ситуации, так как отсутствует история операций, приведших к ней.

Применяемая модель

За основу в работе взята модель, базирующаяся на графе совместного исполнения потоков [1], в которой формально описана процедура построения модели исполнения многопоточного алгоритма на разделяемой памяти в целях определения состояния гонки. Приведем краткое описание модели. В ее основе лежит граф совместного исполнения потоков $G := (V, E)$, в котором путям соответствуют всевозможные варианты исполнения многопоточного алгоритма. Вершинам графа V соответствует множество состояний общей памяти после выполнения очередной атомарной инструкции, дугам E — атомарные операции над разделяемыми переменными. Каждому ребру поставлена в соответствие операция (R — чтение, W — запись и X — другая) и ячейка памяти, над которой осуществляется операция.

Также для модели определена функция корректности, отражающая субъективное понятие о корректном или некорректном исполнении программы. Одна и та же многопоточная программа может быть корректной с точки зрения одной функции корректности и некорректной с точки зрения другой. О процедуре анализа на основании описанной модели скажем ниже. Отметим, что граф G имеет вид ромба, в котором начальная вершина находится вверху, конечная — внизу, а все ребра имеют направление сверху вниз.

Подход взят за основу из-за того, что он имеет малую сложность анализа. Число вершин на каждом из ребер ромба линейно зависит от числа операций, и каждая вершина проходится при анализе только один раз, вследствие чего сложность подхода представляет собой $O(kn)$, где k и n — число операций каждого из потоков.

Немного о сути анализа. Рассмотрим путь на графе из начальной вершины в конечную. При проходе обращаем внимание на то, что два ребра, исходящие из одной вершины, могут оперировать разными ячейками памяти или только читать одну и ту же ячейку. Операции, соответствующие этим ребрам, назовем коммутирующими, а остальные операции назовем не

коммутирующими и обозначим их крестом. То, что две операции разных потоков являются не коммутирующими, говорит о том, что результат алгоритма будет зависеть от порядка их исполнения. В случае если операции коммутируют, для анализа не важен порядок исполнения, поэтому из вершины можно двигаться по любому из исходящих ребер — на содержимое памяти в финальном состоянии это не повлияет. На данном наблюдении основывается построение классов эквивалентности — наборы полных путей на графе G, для которых содержимое ячеек памяти в финальном состоянии одинаково. После нахождения всех классов достаточно взять по одному представителю из каждого класса, чтобы перебрать все возможные последовательности смены ячеек памяти.

В работе [1] вопрос о наличии состояния гонки решается следующим образом. Гонка (в терминах графа G) — это существование двух разных полных путей, для которых в финальном состоянии значения хотя бы одной из ячеек памяти различны. С учетом начальных значений ячеек и того, как их значения меняются, используется метод неопределенных коэффициентов. На каждом ветвлении добавляется коэффициент α к изменению в левой ветви и $(1 - \alpha)$ — в правой. В финальной вершине имеется множество значений, где каждая ячейка памяти в общем случае имеет вид:

$$x_i = f_i(\vec{x}_0, \{\alpha\}),$$

где $\vec{x}_0$ — состояние всех ячеек памяти в начальной вершине, $\{\alpha\}$ — значения неопределенных коэффициентов, f_i — некоторая функция. Исходя из определения понятия гонки и принципов построения и анализа графа, изложенных выше, получаем, что гонка возможна тогда и только тогда, когда

$$\exists i, \{\alpha\}_1, \{\alpha\}_2 : f_i(\vec{x}_0, \{\alpha\}_1) \neq f_i(\vec{x}_0, \{\alpha\}_2).$$

Слабое место подхода заключается в том, что в нем отсутствует привязка к конкретным значениям переменных. В терминах графа G это означает, что, если в графе есть недостижимые дуги, алгоритм не сможет сделать вывод об их недостижимости. Привязка к значениям переменных — критический момент в анализе многопоточных алгоритмов, потому что логика непопадания в критические секции и поведение алгоритмов в целом базируются в реальных задачах именно на значениях переменных. Иначе говоря, в рассмотренном подходе не предусмотрен анализ условных конструкций типа if-else и условий внутри циклов for, while, do-while. Кроме того, нет формальной процедуры анализа алгоритмов, содержащих циклы.

В данной работе предлагается модельный подход, позволяющий не только учитывать попарное сравнение операций потоков, но и анализировать поведение системы в целом, включая определение достижимости ребер и диапазонов значений переменных. Предлагаемый подход включает вычислительную часть и применяется в рамках определенного класса задач.

Методика построения расчетного графа

Общая идея подхода к задаче о нахождении состояния гонки. В качестве исходного материала для анализа служит программный код, исполняемый в нескольких потоках. На основе кода строятся специальные графы, вершинам и ребрам которых соответствует необходимая для анализа информация. Используется два вида графов: граф совместного исполнения потоков и расчетный граф. На основании анализа графа совместного исполнения выбираются пути, для которых возможно состояние гонки. На расчетном графе анализируются только эти пути и выводится результат о наличии состояния гонки.

Построение графа совместного исполнения *n* потоков. Покажем, как строится расчетный граф для двух потоков. Имеем начальную вершину, в которой состояние системы определяется значениями, которыми проинициализированы переменные. Множество ребер E будем обозначать как a_j^i, где i — номер операции первого потока, а j — второго. Тогда каждой вершине можно присвоить два индекса, означающих число операций, сделанных каждым из потоков, чтобы оказаться в этой вершине. Из каждой вершины v_j^i, считая от начальной, будет исходить два ребра: a_j^{i+1} и a_{j+1}^i до тех пор, пока в одном из потоков не закончатся операции — тогда из вершины будет исходить только одно ребро, соответствующее операции другого потока. Построенный таким образом граф, как уже было сказано, имеет вид ромба (рис. 1). Метод по-

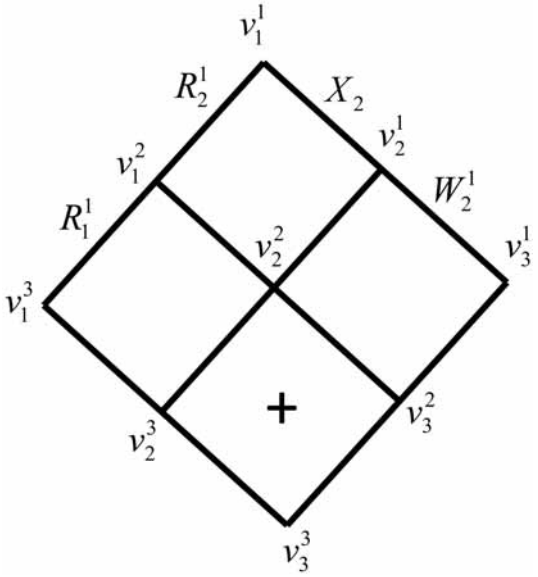


Рис. 1. Пример графа совместного исполнения двух потоков на разделяемой памяти

строения обобщается на случай $n > 2$ потоков: вершины и ребра будут иметь n индексов, а из каждой вершины будет исходить не менее n ребер. В общем случае граф совместного исполнения n потоков будет иметь вид n-мерного ромбовидного графа. Подробное описание примеров и доказательство корректности представления кода в виде графа совместного исполнения потоков даны в работе [1].

Общее представление о расчетном графе. Расчетный граф — конструктивная дискретная модель исходного кода программы, представляющая собой направленный граф, в котором каждому ребру соответствует атомарная операция, а вершинам ставится в соответствие множество значений всех разделяемых переменных. В графе выделена начальная и конечная вершины, соответствующие начальному состоянию переменных перед исполнением и конечному состоянию после исполнения. Проход по графу моделирует один из вариантов совместного исполнения нескольких потоков путем точного указания следования инструкций одна за другой.

Множество значений разделяемых переменных будем называть состоянием системы. Вектор, компонентами которого являются значения разделяемых переменных, назовем вектором состояния. На расчетном графе определим функцию, соотносящую каждой вершине множество значений разделяемых переменных и функцию, соответствующую операции, проводимой над системой на каждом из ребер:

$$S: V \to \vec{x} = (x_1, \ldots, x_n),$$

$$F: E \to g(\vec{x}), \qquad (1)$$

где $\vec{x}$ — состояние системы; g — вектор-функция, показывающая, как меняется состояние системы после прохода по ребру.

Определим на этом графе также функцию условного перехода:

$$M: \left(v_j^i,\ v_j^{i+1}\right) \to P(\vec{x}) = 0, \qquad (2)$$

где P — функция, определенная на множестве значений вектора разделяемых переменных. Это предикат, определяющий возможность передвижения системы из текущей вершины по рассматриваемому ребру, что соответствует условным переходам в программном коде.

Обозначение на рисунках и схемах. Значения всех разделяемых переменных будем обозначать на графе в фигурных скобках: {01}. Если $g(\vec{x})$ меняет значение переменной, то в вершине, куда ребро ведет, новое значение будет подчеркнуто: {0$\underline{1}$}. Каждой атомарной операции соответствует одно ребро (так же, как и в графе совместного исполнения потоков). Отдельно можно выделить специальный вид ребер — ребра условия. Они содержат пустую операцию над переменными: $g \equiv 1$ и функцию-предикат $P(\vec{x})$, определяемую содержанием условия цикла (или содержимым обычного if). Такие ребра появляются, когда в программном коде есть ветвления. Условие прохода по ребру определяет достижимость вершин, куда ребро ведет.

Метод неопределенных коэффициентов. Различным путям графа по построению соответствуют различные варианты исполнения. Находясь в определенной вершине, система может перейти по любому одному из ребер в следующее состояние, что соответствует выполнению инструкции одним из потоков. Для описания ситуации используются неопределенные коэффициенты. Пусть для двух потоков система выполнит инструкцию первого потока с некой вероятностью и значение неопределенного коэффициента будет α, второго — $(1 - \alpha)$. Если в первом потоке с разделяемой переменной совершается операция инкрементации, а во втором — прибавление числа 2, то результатом выполнения такой конструкции будет $\alpha + 2(1 - \alpha) = 2 - \alpha$, где $\alpha = \{0,1\}$.

Представление циклов и ветвлений. Линейный программный код отображается на ребро графа без каких-либо дополнительных действий. Операции чтения соответствует ребро с $g \equiv 1$ и, возможно, нетривиальным предикатом $P(\vec{x})$, если речь идет, например, о конструкции if ($x == 1$). При появлении нелинейных участков, таких как циклы и ветвления, анализ усложняется. Будем использовать для представления ветвлений неопределенные коэффициенты β (аналогично коэффициентам α). Каждой из ветвей приписывается свой коэффициент, и только один из них равен 1, остальные — 0. На выходе имеем выражение с неопределенными коэффициентами, представляющее собой состояние системы на выходе из ветвления. Коэффициенты в выражении определяют ветвь, для которой моделируется исполнение. Обратных ребер в расчетном графе нет. Вместо них у ребер, входящих в тело цикла, появляется зависимость от параметра — номера итерации в цикле. Не всегда зависимость от номера итерации может быть задана явно, однако для реальных задач она, как правило, именно явная. Кроме того, система дополняется условием выхода из цикла. Исходный цикл приобретает форму нескольких однонаправленных ребер, представляющих тело цикла, что, безусловно, облегчает задачу анализа.

Конструктивное построение расчетного графа. Построение расчетного графа описано выше в разделе "Построение графа совместного исполнения n потоков". Отличия от графа совместного исполнения потоков заключаются в данных, сопоставляемых вершинам и ребрам. Каждому ребру соотносят функции (1) и (2). Для определенности будем подразделять ребра на два вида: ребра условия, где P — нетривиальна, g — тривиальна и ребра операций, где P — тривиальна, g — нетривиальна. Проход по такому графу из начальной вершины в конечную однозначно определяет последовательность взаимного выполнения инструкций двух потоков. Построение расчетного графа для n потоков полностью аналогично.

Расчеты на графе

После того как расчетный граф, состоящий из ребер операций и ребер условий, построен, и начальной вершине соотнесено множество исходных значений разделяемых переменных, можно приступать к расчетам. Каждая из итераций — проход по одному из путей из начальной вершины в конечную. В общем случае число путей достаточно велико — C_n^k. Однако будем рассматривать только те пути, которые были отобраны с помощью классов эквивалентности на графе совместного исполнения потоков. Таких путей значительно меньше — $O(nk)$. Элементарная часть итерации — проход из текущей вершины по ребру в следующую в соответствии с выбранным путем и затем либо изменение вектора состояния, либо проверка условия достижимости и проход по ребру (или признание ребра недостижимым). Если в графе были ветвления, то ребра, соответствующие ветвям цикла, будут иметь коэффициент β и $(1 - \beta)$. Коэффициенты войдут в выражение для значений переменных в финальной вершине. В зависимости от рода задачи и функции корректности по графу делается вывод о недостижимости критической секции или присутствии состояния гонки для конкретной разделяемой переменной.

Рассмотрим анализ алгоритма Петерсона в качестве иллюстрации применения предлагаемого подхода (рис. 2). Внутренние ребра намеренно не подписаны, но подразумевается, что расчеты проводятся и для них тоже. В фигурных скобках указаны значения разделяемых переменных. Подчеркнуты те значения, которые изменились при переходе по ребру, а также начальные значения переменных. Также подписаны ребра условий, например: "$y = 0 \parallel z = 0$". Внутренние ребра, отсутствующие на рисунке, недостижимы.

Рассмотрим путь, выделенный полужирными линиями, и вершину A на нем.

Может ли система пойти из этой вершины по правому ребру? В вершине F значение вектора состояния {110}. Условие попадания из F в A — выполнение предиката $y = 0 \parallel z = 0$. Если предикат верен, ребро, входящее в A, достижимо. Достижимость ребра AS определяется предикатом $x = 0 \parallel z = 1$. При векторе состояния {110} значение предиката ложно, поэтому ребро недостижимо.

Доказательство корректности подхода

Ранее было сделано довольно смелое утверждение о том, что для анализа программы в расчетном графе можно использовать пути, выявленные с помощью графа совместного исполнения потоков. Докажем, что утверждение верно.

Прежде чем приступить к формулировке и доказательству соответствующей теоремы, обратим внимание на некоторые свойства классов эквивалентности.

Лемма. У путей, принадлежащих одному классу эквивалентности, по каждой разделяемой переменной зафиксирован порядок операций.

Это означает, что если взять все множество последовательностей операций, соответствующих всем путям одного класса эквивалентности, выбрать любую из разделяемых переменных и посмотреть на порядок следования операций, совершаемых с этой переменной, то окажется, что порядок операций с этой переменной всегда одинаков.

Доказательство. Рассмотрим произвольный граф совместного исполнения потоков и произвольный класс эквивалентности этого графа. На рис. 3 приве-

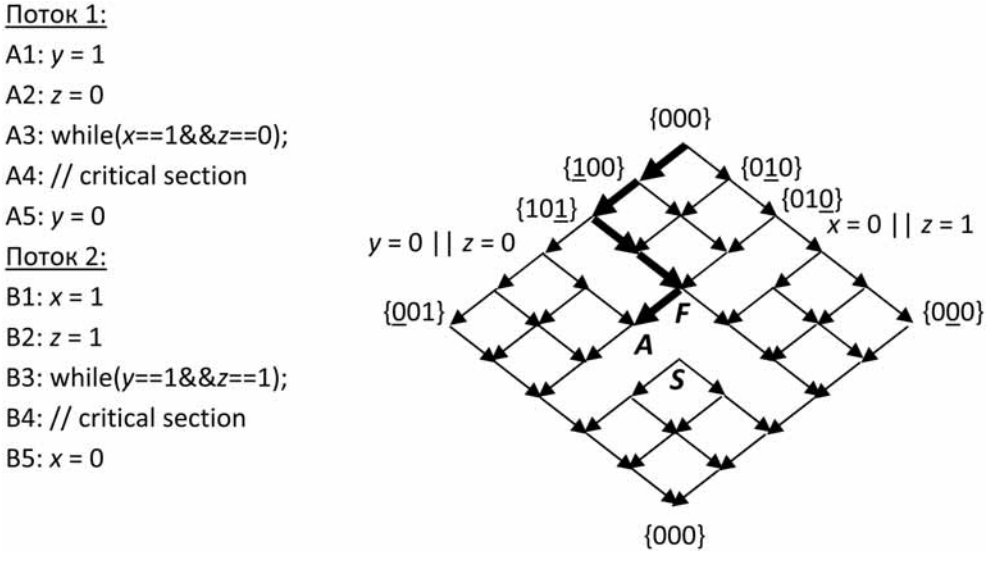


Рис. 2. Расчетный граф для алгоритма Петерсона

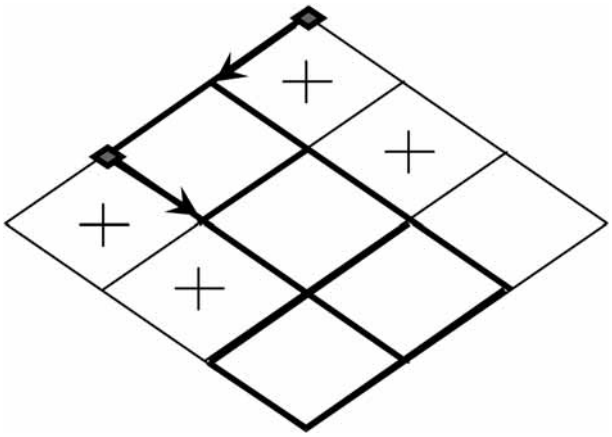


Рис. 3. Положение класса эквивалентности относительно коммутирующих ячеек

ден пример такого графа. Знаком "+" обозначены ячейки графа, образованные ребрами с не коммутирующими операциями, полужирными линиями — класс эквивалентности. Доказывать будем от противного.

Выберем класс эквивалентности, два пути на нем и некоторую разделяемую переменную.

Предположение: допустим, порядок операций с этой переменной для выбранных путей различный.

Обратим внимание, что в конечной вершине все операции так или иначе будут выполнены, значит, имеет место перестановка операций чтения/записи. Значит, состоялась одна из трех перестановок:

1. $R \leftrightarrow R$;
2. $W \leftrightarrow R$;
3. $W \leftrightarrow W$.

Первый вариант нас не интересует, так как в обоих потоках происходит считывание значения переменной, и случай для нас неразличим. Рассмотрим перестановки 2 и 3. В них очередность исполнения явно влияет на содержимое разделяемой переменной. Обратим внимание, что местами могут меняться только операции из разных потоков, внутри одного потока порядок зафиксирован ("порядок выполнения программы"). Значит, существует ячейка с ребрами W и W/R, каждое из которых входит в состав класса эквивалентности. Но такая ячейка является не коммутирующей, что исключает возможность ее принадлежности к одному классу эквивалентности. Предположение не верно, порядок следования операций — одинаковый.

Теорема. Для того чтобы сделать вывод о наличии или отсутствии состояния гонки, *достаточно* рассмотреть представителей классов эквивалентности из графа совместного исполнения потоков.

Доказательство. При наличии единственной разделяемой переменной утверждение теоремы прямо следует из доказанной леммы. В ситуации, когда есть несколько разделяемых переменных, утверждение теоремы не очевидно и требует дополнительного доказательства. Далее предлагается доказательство от противного.

Пусть существуют два пути, принадлежащие одному классу эквивалентности, приводящие к различным значениям разделяемых переменных.

Допущение сформулировано, используя отрицание формулировки теоремы и исходя из определения понятия гонки (данного во введении).

Из различия значений переменных после исполнения следует, что существует некая строчка исходного кода, действия в которой приводят к разным значениям одной из переменных. Это может быть либо явное присваивание переменной нового значения, например:

```
shared_x = func(shared_y, shared_z);
```

либо логическое ветвление кода, приводящее к той же записи в переменную:

```
if (shared_y = = 0)
        shared_x = 1;
else
        shared_x = 0.
```

В любом из случаев на значение разделяемой переменной влияют значения остальных разделяемых переменных.

Уточнение: если на ветвление влияют неопределенные в рамках подхода факторы, например: if (rnd() > 0.3), то рассматриваются все возможные ветви и если хоть в одной из них возникнет ситуация гонки, то общим ответом будет, что ситуация гонки возможна.

Вернемся к сформулированному выше допущению: предполагаем, что проход по разным путям в графе приводит к различным значениям переменных после исполнения программы. Отметим, что исходные значения разделяемых переменных не зависят от пути на графе совместного исполнения потоков. К интересующему нас моменту ветвления или записи значения переменных одинаковы для обоих потоков. Это следует из доказанной ранее леммы для каждой из переменных и из того факта, что до этого никаких ветвлений и записи не было.

Уточнение: если ранее были ветвления или запись, то повторим доказательство для этого предыдущего ветвления или записи.

Но если значения всех переменных к моменту ветвления или записи одинаковы, то и результат соответствующих операций сравнения и записи будет одинаковым, а значит предположение о том, что для разных путей на графе в результате исполнения получаются разные значения, неверно.

Обоснованность использования представителей класса эквивалентности в расчетном графе. Воспользовавшись доказанной теоремой, получаем одинаковую последовательность операций с разделяемыми

переменными в рамках каждого класса эквивалентности. Можно рассматривать это так, что каждый из потоков получает управление строго в определенный момент, когда это повлияет на дальнейшее исполнение программы, и до момента следующей развилки исполняется "в одиночестве", и так до момента выхода на не коммутирующие ячейки. Такие конфликты исполнения в терминах графа совместного исполнения потоков описываются как не коммутирующие ячейки. Таким образом, сложность анализа программного кода можно существенно сократить, взяв по одному пути из каждого класса эквивалентности.

Заключение

Предложен и обоснован новый подход к проблеме поиска состояний гонки. Подход относится к классу статического анализа и обладает меньшей сложностью анализа, чем его известные аналоги. Сложность существенно уменьшена путем классификации части возможных вариантов исполнения как эквивалентных с определенной точки зрения. Обоснованность сокращения вариантов исполнения аналитически доказана.

Формализм и идея подхода могут служить в качестве базы для создания программного средства статистического анализа многопоточных алгоритмов.

Список литературы

1. **Кудрин М. Ю., Прокопенко А. С., Тормасов А. Г.** Метод нахождения состояний гонки в потоках, работающих на разделяемой памяти // Труды МФТИ. 2009. Т. 1. № 4. С. 182—201.

2. **Serebryany K.** Data race test. URL: http://code.google.com/p/data-race-test

3. **Каличкин С. В.** Обзор средств статической отладки. Новосибирск: Прайс-курьер, 2004. С. 22.

4. **Карпов А.** Тестирование параллельных программ. URL: http://www.software-testing.ru/library/testing/functional-testing/581-parallelprogramtesting

5. **Калугин А.** Верификатор программ на языке C LINT. URL: http://www.viva64.com/go.php?url=224

6. **Intel** Architecture Soft. Dev.'s Manual. Vol. 1.

7. **Emanuelsson P., Nilsson U.** A Comparative Study of Industrial Static Analysis Tools. Amsterdam: Elsevier Science Publishers, 2008.

8. **Static** Source Code Analysis Tools for C. URL: http://www.spinroot.com/static/

9. **Использование** Thread Analyzer для поиска конфликтов доступа к данным. URL: http://ru.sun.com/developers/sunstudio/articles/tha_using_ru.html.

10. **Herlihy M., Luchangco V., Moir M.** Obstruction-free synchronization: Doubleended queues as an example // International Conference on Distributed Computing Systems. Providence. RI. USA 2003. P. 522—529.

11. **Herlihy M., Shavit N.** The Art of Multi processor Programming. Amsterdam: Elsevier Science Publishers, 2008.

ИНФОРМАЦИЯ

УДК 004.75; 004.415, 004.946

В. Ю. Харитонов, канд. техн. наук, мл. науч. сотр., Московский энергетический институт (технический университет),
e-mail: kharitonovvy@gmail.com

Инструментальные программные средства для построения распределенных систем виртуальной реальности. Часть II*

Статья посвящена важной и актуальной в настоящее время проблеме создания распределенных систем виртуальной реальности. Предлагается комплексный подход, основанный на разработке специализированных инструментальных программных средств, призванных упростить и одновременно ускорить процесс создания конечной системы.

Во второй части статьи рассматриваются механизмы, обеспечивающие согласованность данных в предлагаемой программной архитектуре распределенных систем виртуальной реальности, а также основные возможности и детали реализации программного комплекса TerraNet.

Ключевые слова: *распределенные системы виртуальной реальности, обеспечение согласованности данных, распределенное моделирование, вычислительные сети, трехмерная интерактивная компьютерная графика*

Разработка распределенных систем виртуальной реальности (РСВР) является одним из наиболее перспективных направлений развития современных распределенных систем. Такие системы состоят из множества территориально-распределенных *компонентов*, взаимодействующих друг с другом в режиме реального времени в целях создания общей для множества участников виртуальной среды. В зависимости от типа РСВР эти компоненты могут выполнять различные задачи. Например, в специализированных РСВР, ориентированных на промышленное применение, таких как современные тренажерные системы, компоненты могут решать задачи, связанные с визуализацией виртуальной среды, моделированием реальных физических условий, а также с организацией человеко-машинного интерфейса. В массовых РСВР, разрабатываемых, в основном, для индустрии развлечений, таких как сетевые компьютерные игры, компонент РСВР может ассоциироваться с отдельным игроком, обеспечивая его доступ в игровое пространство и взаимодействие с другими игроками. В независимости от типа системы, обеспечение согласованного взаимодействия компонентов РСВР требует применения комплексного подхода, предусматривающего разработку специализированного программного обеспечения (ПО). В данной статье предлагается один из вариантов организации такого ПО — программный комплекс TerraNet, представляющий собой ПО промежуточного уровня для интеграции компонентов современных РСВР и обеспечивающий их взаимодействие в реальном масштабе времени.

Программный комплекс TerraNet (далее — библиотека TerraNet) позволяет избежать низкоуровневого сетевого программирования и, тем самым, значительно упростить процесс разработки РСВР, предоставляя высокоуровневый программный интерфейс для создания, управления и распределения объектов в рамках единой виртуальной среды. Встроенные механизмы управления репликацией данных гарантируют согласованность данных и предоставляют средства для динамической балансировки загрузки сети. Меха-

* Часть I опубликована в журнале "Программная инженерия" № 6, 2011.

низм *взаимодействий* позволяет разработчику задавать собственные алгоритмы взаимодействия процессов РСВР. Кроме того, благодаря тому, что ядро библиотеки TerraNet отделено от графической составляющей, у разработчика появляется возможность использовать данную библиотеку совместно с любой внешней системой визуализации.

Во второй части статьи рассматриваются механизмы, обеспечивающие согласованность данных в рамках описанной в первой части программной архитектуры. Кроме того, приводятся детали реализации, основные возможности и примеры работы с программным комплексом TerraNet.

1. Механизмы обеспечения согласованности данных

1.1. Принцип "избирательной согласованности"

В зависимости от типа, а также индивидуальных особенностей объектов виртуальной среды, их состояния могут включать множество различных атрибутов. Так, состояние статического объекта может содержать координаты объекта, параметры, описывающие его размеры, форму, физические характеристики и т. д. Состояние динамического объекта дополнительно может включать скорость, ускорение, значения сил и моментов. Соответственно, для обеспечения полностью согласованного взаимодействия пользователей необходима согласованность по всем этим параметрам.

Поскольку взаимодействие пользователей в РСВР осуществляется в реальном времени, то моделирование и особенно визуализация состояния виртуальной среды должны проводиться с высокой частотой. Например, по современным стандартам частота визуализации в тренажерных комплексах не должна опускаться ниже 60 Гц. Пересылка полных состояний объектов с такой частотой потребовала бы очень высокой пропускной способности каналов передачи данных — такой подход оказывается накладным. Кроме того, полное состояние объекта часто является избыточным и необходимо лишь на стороне процесса — источника данных, непосредственно управляющего объектом, для точного моделирования его состояния. В большинстве случаев для остальных процессов нет необходимости полностью моделировать состояние объекта, достаточно лишь поддерживать некоторую его упрощенную модель. Соответственно, для них не нужно передавать специфические характеристики объекта. Например, для динамических объектов можно не передавать значения сил и моментов, ограничившись лишь скоростью и ускорением.

Основная идея принципа "избирательной согласованности" — выделить наиболее значимые для конкретной задачи атрибуты состояния объекта и поддерживать согласованность только по ним. Данные атрибуты могут быть выбраны на основе различных соображений, среди которых наиболее важное — человеческое восприятие. Таким образом, принцип допускает, что между локальными состояниями процессов пользователей могут быть различия, но они являются малозаметными для восприятия. В результате, в соответствии с выражением (4) из первой части статьи (см. журнал "Программная инженерия", № 6, 2011) появляется возможность найти компромисс между согласованностью и чувствительностью.

Главные отличительные особенности принципа "избирательной согласованности":

а) динамический выбор параметров состояния объектов виртуальной среды, подлежащих репликации;

б) управление частотой репликации атрибутов на основе задаваемых стратегий;

в) использование "дельта-компрессии" при сериализации объектов;

г) применение упрощенных моделей объектов на процессах-приемниках данных при одновременном использовании методов предсказания состояний объектов [1].

1.2. Стратегии репликации данных и дельта-компрессия

Высокоуровневый протокол предоставляет пользователю возможность гибкого управления репликацией атрибутов состояний объектов при изменении их значений. Пользователь может выбирать между четырьмя основными стратегиями репликации:

- *репликация при каждом локальном изменении значения атрибута* (стратегия *Replicate-on-Every-Change — REC*) — изменение значения атрибута на каком-либо процессе ведет к немедленной его репликации на другие процессы системы;
- *периодическая репликация с заданной частотой* (стратегия *Fixed-Rate-Replication — FRR*) − значение атрибута реплицируется с заданной разработчиком частотой;
- *явная репликация* (стратегия *Manual Replication — MR*) — репликация значения атрибута проводится только при явном указании (вызове метода Serialize() у атрибута);
- *репликация по условию* — с переменной частотой (стратегия *Conditional-Replication — CR*) — разработчик назначает на атрибут логическое выражение (условие), используемое для определения необходимости репликации его значения.

Перечисленные стратегии позволяют более эффективно использовать пропускную способность сети. Среди всех стратегий репликации наилучшую согласованность данных обеспечивает стратегия *REC*. Она же характеризуется наибольшей загрузкой сети. Стратегия *FRR* обеспечивает приемлемый уровень согласованности и более равномерную загрузку сети за счет того, что не все изменения значений атрибутов передаются по сети. Две другие стратегии репликации (*MR* и *CR*) являются наиболее гибкими и предоставляют разработчику возможность самостоятельно задавать алгоритм репликации.

Дельта-компрессия (положение *в* Принципа) гласит о том, что в каждом цикле моделирования по сети передаются не полные состояния объектов виртуальной среды, а только лишь их измененные части (так называемые *дельты* состояний). Использование дельта-компрессии является очевидным и в то же время чрезвычайно выгодным, поскольку позволяет значительно экономить пропускную способность сети без каких-либо потерь в согласованности данных.

Дельта-компрессия работает следующим образом. При изменении значения атрибута, он, в соответствии с назначенной стратегией репликации, может быть помечен как требующий репликации. Все атрибуты состояния объекта, помеченные в данном цикле моделирования как требующие репликации, упаковываются в сообщение с идентификатором `NET_MSG_OBJECT_SET_STATE_DELTA` и за один вызов *Send()* отправляются по сети.

1.3. Предсказание состояний объектов виртуальной среды

Предсказание состояний объектов (*dead reckoning* — положение *г* Принципа) позволяет уменьшить количество передаваемых внутри РСВР данных и тем самым повысить масштабируемость системы. Идея, лежащая в основе метода, — реже передавать сообщения обновления, при этом используя содержащуюся в них информацию для расчета недостающих состояний объекта. Иллюстрация работы предсказания состояния объекта для простейшего линейного случая приведена на рис. 1. На верхнем графике показана траектория движения объекта, управляемого пользователем U_A, а на нижнем — траектория движения того же объекта, видимая удаленным пользователем U_B, использующим метод предсказания состояния (жирная серая линия). Пользователь U_A передает состояние своего объекта, включающее текущее положение объекта $r(t_i)$ и скорость $v(t_i)$, пользователю U_B в моменты времени t_i, а пользователь U_B принимает его в моменты времени t_i и использует в дальнейшем для линейной экстраполяции состояния объекта.

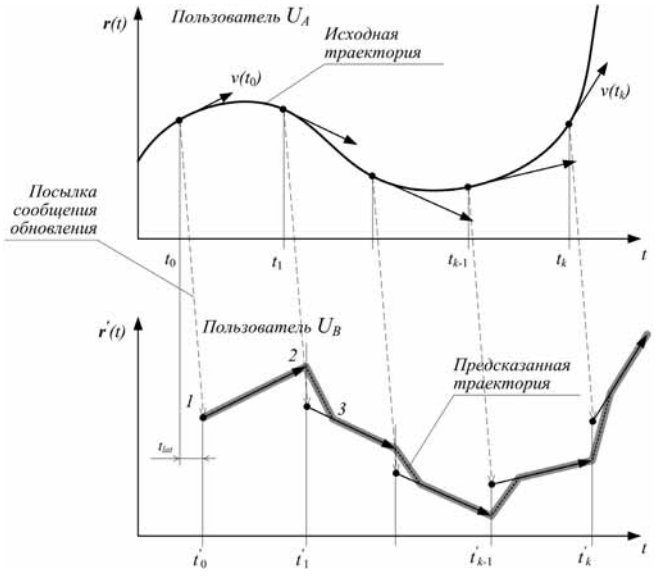


Рис. 1. Иллюстрация работы предсказания состояния объекта для линейного случая

Применение метода предсказания подразумевает решение двух вопросов: выбор частоты посылки данных на стороне источника данных и выбор алгоритма предсказания на принимающей стороне. Оба эти вопроса взаимосвязаны: частота посылки данных должна быть выбрана так, чтобы избранный алгоритм предсказания позволял сохранить требуемую согласованность между пользователями при минимальном потребляемом сетевом трафике.

Наиболее перспективными в настоящее время представляются адаптивные методы предсказания [2, 3], позволяющие динамически варьировать частоту посылки сообщений обновления в соответствии с какими-либо критериями.

В реализованном в библиотеке TerraNet адаптивном методе предлагается варьировать не только частоту посылки сообщений обновления, но и сам алгоритм предсказания. Главным критерием смены алгоритма предсказания является тип движения объекта (прямолинейное равномерное движение, прямолинейное равноускоренное движение, криволинейное движение и т. п.). При изменении типа движения сторона, непосредственно управляющая объектом, информирует об этом принимающую сторону, в результате чего та начинает применять другой тип алгоритма предсказания. Данный подход позволяет лучше предсказывать динамику поведения объекта, движущегося по сложной траектории. Кроме того, появляется возможность динамически подстраивать размер сообщения обновления под конкретный тип движения объекта, что позволяет более гибко управлять сетевым трафиком.

1.4. Управление совместным доступом к состоянию виртуальной среды

При построении РСВР очень важно правильно обрабатывать обращения нескольких пользователей к одним и тем же данным. Такая ситуация часто встречается, например, когда несколько пользователей пытаются одновременно управлять одним объектом. Действия разных пользователей, пересекающиеся во времени, не должны конфликтовать друг с другом. Чтобы обеспечить это свойство, необходим специальный механизм контроля доступа к состояниям объектов (*ownership management*).

В библиотеке TerraNet применен подход, основанный на передаче права владения объектом. Каждый объект виртуальной среды имеет своего владельца. Процесс, владеющий объектом, вправе изменять его состояние и взаимосвязи с другими объектами. При этом право владения объектом может быть передано от одного процесса другому. Процесс, желающий за-

владеть объектом, должен послать соответствующий запрос серверу. Если запрашиваемый объект является свободным (т. е. не захвачен другим процессом), то сервер помечает в глобальном хранилище указанный объект как занятый и отсылает процессу положительный ответ. После этого изменять состояние объекта в глобальном хранилище может только процесс-владелец объекта. Если же требуемый объект захвачен другим процессом, сервер запрашивает право на владение этим объектом у этого процесса. Процесс-владелец может как принять данный запрос и передать право владения, так и отклонить запрос, в зависимости от своих нужд. Кроме того, он имеет право заблокировать доступ к объекту. В этом случае любой запрос на право владения объектом будет автоматически отклоняться.

2. Работа с программным комплексом TerraNet

По данным одного из последних исследований [4], наиболее перспективным направлением исследований в области создания РСВР является разработка программного обеспечения промежуточного уровня (*middleware*). ПО промежуточного уровня представляет собой прослойку между компонентом РСВР и операционной системой с ее низкоуровневыми сетевыми программными интерфейсами, которая обеспечивает прозрачное взаимодействие данного компонента с другими компонентами РСВР. Такое ПО позволяет абстрагироваться от разработки внутренних сетевых механизмов функционирования РСВР и сосредоточиться на процессе построения самой виртуальной среды.

Основной целью при создании библиотеки TerraNet была реализация мощного инструмента, предоставляющего разработчику возможность взаимодействия с РСВР на пользовательском уровне абстракции, т. е. на уровне отдельных объектов виртуальной среды, при этом скрыв от него все низкоуровневые сетевые механизмы. Программный интерфейс библиотеки практически не содержит функций передачи данных, а включает лишь прототипы основных сущностей, из которых, как из конструктора, можно синтезировать требуемую конфигурацию виртуального мира.

2.1. Обзор возможностей

Основу программного комплекса TerraNet составляют несколько модулей, представляющих собой динамически связываемые библиотеки (*Dynamic Link Library*, DLL):

- *terranet_core* — ядро комплекса, включающее в себя весь функционал, необходимый для развертывания сетевой инфраструктуры РСВР;
- *terranet_osg* — графическая составляющая, включающая встроенную оконную систему и систему визуализации на основе графа сцены;
- *network* — сетевая составляющая, реализующая базовые сетевые функции, необходимые для функционирования ядра (в последней версии библиотеки включена в состав ядра).

Структура библиотеки и взаимосвязь ее модулей между собой, приложением пользователя (компонентом РСВР) и компонентами операционной системы (ОС) показана на рис. 2.

Перечислим основные возможности библиотеки TerraNet:

- поддержка иерархических взаимосвязей между объектами виртуальной среды (представление виртуальной среды в виде распределенного графа сцены), работа с объектами виртуальной среды на уровне атрибутов их состояний;
- возможность назначения на объекты различных стратегий репликации данных и методов предсказания состояний объектов, в том числе предложенных самим разработчиком;
- распространение обновлений состояния виртуальной среды на основе механизма подписки, поддержка множественной рассылки;
- возможность написания собственных контроллеров управления состоянием объектов — контроллеры подключаются как внешние модули и назначаются на объекты;
- наличие механизмов управления совместным доступом к состоянию виртуальной среды многих пользователей;
- ядро библиотеки отделено от графической составляющей — благодаря этому разработчик может использовать данную библиотеку вместе с любой другой системой визуализации;
- платформонезависимость — исходный код библиотеки написан на стандартном языке C++ с использованием кроссплатформенных библиотек *Sockets, OpenGL* и *OpenSceneGraph* [5].

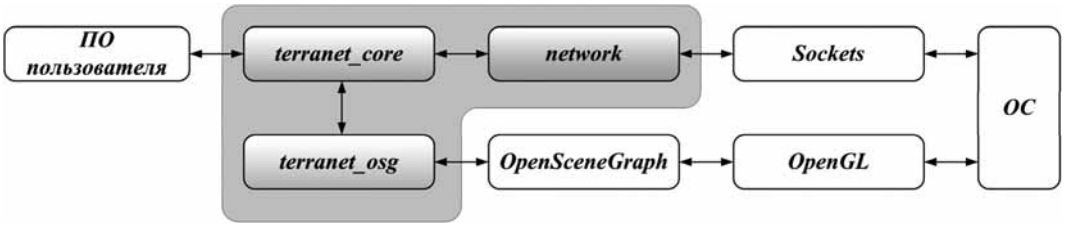


Рис. 2. Структура библиотеки TerraNet и взаимосвязь ее модулей

2.2. Интерфейсные классы ядра

В основу системы классов ядра была положена широко известная концепция *модель-вид-контроллер* (*Model-View-Controller — MVC*) [6], позволяющая выделить три отдельных составляющих программного приложения — *модель данных*, *графическое представление* и *управляющую логику*, так, что модификация одной из них оказывает минимальное воздействие на другие.

К классам модели в концепции *MVC* относятся интерфейсные классы *TA_World*, *TA_Object*, *TA_State, TA_StateAttribute*. Как упоминалось в первой части статьи, все объекты виртуальной среды группируются в *граф сцены*, распределенный между всеми процессами системы. Класс *TA_World* является контейнером, хранящим граф сцены, класс *TA_Object* описывает отдельный узел графа — объект виртуальной среды.

Классы *TA_State* и *TA_StateAttribute* служат для работы с состоянием объекта на уровне отдельных атрибутов. Пользователь может создавать/удалять атрибуты различных типов, изменять их значения, назначать стратегии репликации и т. д.

К группе классов управляющей логики относятся контроллер *TA_Controller* и его производные классы. Основная задача контроллеров — задание законов изменения состояний объектов. Например, в случае применения в авиационных тренажерах контроллер может реализовывать динамику движения реактивного истребителя. В более общем смысле контроллер позволяет разработчику задавать произвольный алгоритм обработки состояния объекта. Так, контроллер может реализовывать один из методов предсказания состояния объекта или использоваться для сбора статистики об объектах.

Библиотека TerraNet предоставляет централизованный механизм управления ресурсами — *менеджер ресурсов*, доступный через класс *TA_ResourceManager*. Почти все ресурсы в TerraNet создаются с помощью менеджера ресурсов, который представляет собой объект-синглетон [6] и, таким образом, доступен из любой точки приложения пользователя. При этом пользователю не нужно заботиться об освобождении созданных им ресурсов — все объекты автоматически удаляются по завершении работы программы.

2.3. Написание приложений с помощью TerraNet API

Написание любого приложения с использованием TerraNet API (*TerraNet Application Programming Interface*) начинается с инициализации, в процессе которой определяется роль текущего процесса в схеме взаимодействия РСВР (клиент либо сервер), проводится инициализация сетевого компонента, инициализируются и подключаются (если это необходимо) графические компоненты, строится граф сцены, проводится синхронизация времени и выполняются другие необходимые операции. В следующем фрагменте кода приведен пример инициализации клиентского приложения:

```
// Получаем указатель на менеджер ресурсов
TA_ResourceManager* pResMan =
   TA_ResourceManager::GetInstance();

// Инициализуем текущий процесс как клиентское
// приложение и сообщаем ядру, что для
// визуализации необходимо использовать
// OpenSceneGraph
pResMan->Init(TA_CLIENT,
    new OSGResourceManagerImpl);

// Получаем доступ к интерфейсу
// межпроцессорного взаимодействия
TA_NetworkInterface* pNetInterface =
   pResMan->GetNetworkInterface ();

// Подключаемся к серверу и включаем
// поддержку множественной рассылки
if (!pNetInterface->Connect
   ("192.168.1.1", true))
   return false;

// Проводим синхронизацию локальных часов
// с часами сервера
pResMan->GetTimer()->Sync();

// Получаем доступ к локальному хранилищу
// виртуальной среды
TA_World* pWorld = pResMan->GetWorld();

// Создаем камеру и окно для визуализации
// (опционально)
// Получаем доступ к системе визуализации
// (опционально)
pRenderer = pResMan->GetRenderer();

// Связываем систему визуализации с данными
// для отображения (состоянием виртуальной
// среды)
pRenderer->SetSceneData(pWorld);

// Получаем доступ к менеджеру обработки
// ввода
TA_InputManager* pInputManager =
   pResMan->GetInputManager();
```

После вызова у сетевого интерфейса метода *Connect()* в соответствии с протоколом *DVRP* клиенту присваивается уникальный идентификатор и передается текущее состояние виртуальной среды. Синхронизация часов завершает инициализацию приложения. Единственное что остается сделать — это запус-

тить основной цикл приложения, который выглядит следующим образом:

```
while (pInputManager->GetKey()
!= TA_KEY_ESCAPE)
{

   // Обработка текущего состояния
   // виртуальной среды
   pWorld->Simulate();

    // Визуализация вида виртуальной среды
    pRenderer->RenderFrame();
}
```

2.4. Операции над виртуальной средой

Среди основных операций, предоставляемых TerraNet API, наиболее важными являются операции над виртуальной средой. К ним относятся:

— операции по созданию и удалению объектов виртуальной среды;

— операции по связыванию объектов между собой в граф сцены;

— операции по модификации состояний объектов.

Создание объекта подразумевает его регистрацию в глобальном хранилище состояния виртуальной среды с последующим назначением уникального идентификатора и имени, под которыми он будет доступен для всех процессов РСВР. При этом если пользователь желает связать с объектом некоторое графиче-

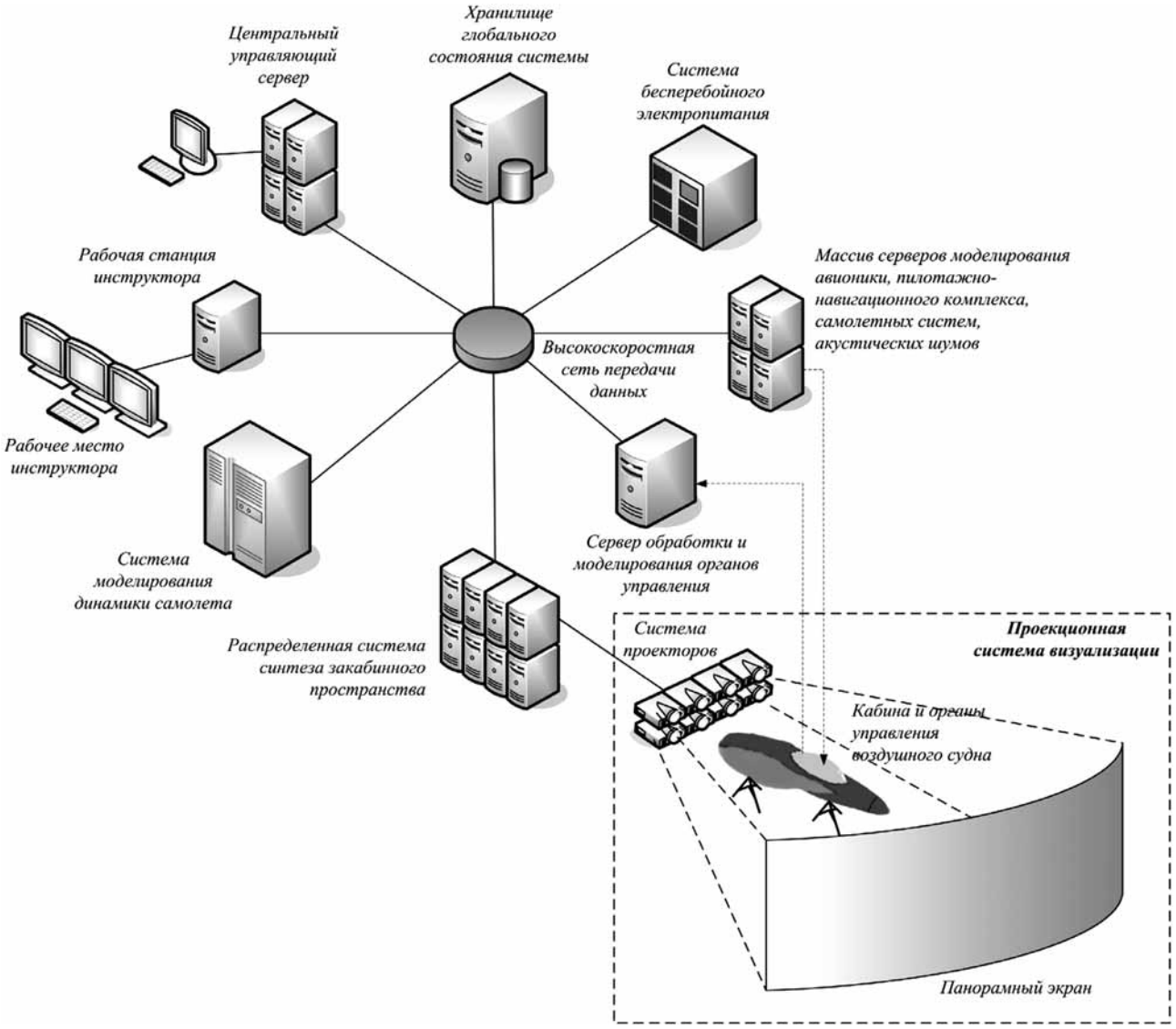


Рис. 5. Структурная схема комплексного авиационного тренажера

ское представление, он дополнительно должен создать модель *TA_Model* и связать ее с объектом. Для того чтобы, к примеру, создать и отобразить объект "Самолет", пользователю необходимо написать следующий фрагмент кода:

```
// Создание и автоматическая репликация
// объекта
TA_Object* pAirplane =
   pResMan->CreateObject("Airplane");

// Связывание объекта с графическим
// представлением
TA_Model* pModel =
   pResMan->CreateModel("aiplane.3ds")
pAirplane->SetModel(pModel);
```

Связывание объектов в граф сцены проводится по иерархическому принципу с использованием отношения "родитель-потомок". Добавление потомков к объекту осуществляется посредством вызова метода *AddChild()*. С каждым объектом может быть связано произвольное число объектов-потомков.

Модификация состояния объекта осуществляется через интерфейсный класс *TA_State*. Под модификацией состояния понимается добавление/удаление атрибутов и изменение их значений. Как это делается рассмотрим на следующем примере. Пусть нам необходимо добавить в виртуальную среду объект "Аэродром", а затем установить на него созданный ранее самолет в определенное положение и ориентацию. Для этого нужно добавить следующий фрагмент кода:

```
// Создаем объект "Аэродром"
TA_Object* pAirfield =
   pResMan->CreateObject("Airfield");
pAirfield->SetModel(pResMan->CreateModel
   ("airfield.3ds"));

// Связываем объект "Самолет" с объектом
// "Аэродром"
pAirfield->AddChild(pAirplane);

// Связываем объект "Аэродром"
// с корневым узлом графа сцены
pWorld->GetRootObject()->AddChild
   (pAirfield);

// Модифицируем состояние объекта "Самолет",
// задавая его положение относительно
// аэродрома
TA_State* pState = pAirplane->GetState();
TA_StateAttribute* pAttr =
   pState->GetAttribute("POSITION");
pAttr->SetValue(TA_Point3D(20.0, 30.0, 0.0));
```

Заметим, что в рассмотренных примерах кода нет ни одной сетевой операции — при проектировании виртуальной среды пользователь работает исключительно с ее объектами. При этом процесс создания графа сцены будет автоматически транслироваться всем остальным подключенным к системе клиентам, и все удаленные пользователи смогут увидеть созданные объекты.

2.5. Подписка на обновления состояний объектов

Как говорилось ранее, для распространения сообщений обновления в библиотеке TerraNet применяется механизм подписки. По умолчанию любой клиентский процесс получает информацию о состоянии всех объектов, существующих в виртуальной среде. Однако в рамках РСВР существует возможность организации так называемой "подписки" на обновления от конкретных объектов или обновления от конкретных типов объектов (называемых *шаблонами состояний*). Для этого в классе *TA_NetworkInterface* существуют соответствующие методы: *SubscribeObject()* и *SubscribePattern()*. При вызове любого из этих методов "область видимости" вызывающего процесса сужается до конкретных объектов или типов объектов. Причем подписка на шаблон состояния является более общим случаем подписки на отдельный объект: процесс принимает сообщения обновления от всех объектов, созданных по определенному шаблону.

По умолчанию, механизм подписки реализован через создание/удаление определенных multicast-групп. Тем не менее, не все клиенты могут поддерживать технологию множественной рассылки. Для них предусмотрен специальный режим эмуляции подписки на сервере.

2.6. Взаимодействие между процессами

Помимо внутренних механизмов взаимодействия процессов, составляющих высокоуровневый протокол и скрытых от пользователя, в интерфейсе библиотеки TerraNet предусмотрен механизм для реализации пользовательских алгоритмов межпроцессного взаимодействия. Пользователь имеет возможность регистрировать в распределенной системе и распространять между процессами *взаимодействия* (*interactions*), представляющие собой сообщения специального вида с произвольным набором параметров. За обработку входящих взаимодействий на процессах РСВР отвечают специальные обработчики, которые пользователь также определяет самостоятельно. Для взаимодействий также применяется механизм подписки — процессы РСВР могут подписываться только на интересующие их взаимодействия. В результате можно реализовать произвольный протокол межпроцессного взаимодействия.

2.7. Вспомогательные средства

В состав библиотеки TerraNet входит набор служебных средств для мониторинга состояния РСВР и сбора сетевой статистики. С любого процесса РСВР в реальном времени можно получать информацию о количестве, статусе и подписках подключенных клиентов; запрашивать сетевые показатели, такие как средний входящий трафик и время сетевого отклика от

любого интересующего процесса. Все эти средства доступны через интерфейсный класс *TA_TetworkInterface*.

Другой интересной возможностью является возможность записи треков состояний объектов виртуальной среды. Пользователь может выбрать удаленный процесс, на котором будет проводиться запись; атрибуты, по которым будет проводиться запись; а также задать время начала записи и общую ее продолжительность. Для этого у класса *TA_Object* предусмотрен метод *RecordStateTrack()*. Записанные таким образом *треки состояния* могут быть затем воспроизведены в любой момент времени и на любом процессе с помощью метода *PlayStateTrack()*. Треки состояния являются чрезвычайно полезными при анализе производительности РСВР и связанным с ним расчетом согласованности данных на различных процессах [7].

3. Приложения и направления дальнейших исследований

Библиотека TerraNet может быть использована для различных приложений, требующих взаимодействия множества компонентов в реальном времени. На ее основе было разработано несколько экспериментальных версий РСВР:

- программный комплекс *FFSystem* (*Formation Flight System*), обеспечивающий модельное воспроизведение группового пилотажа нескольких летательных аппаратов над местностью [8] (рис. 3, см. третью сторону обложки);
- система распределенного моделирования виртуальных трехмерных сцен *ShareEdit*, позволяющая нескольким пользователям осуществлять совместное геометрическое моделирование произвольных трехмерных сцен в реальном времени (рис. 4, см. третью сторону обложки).

В настоящее время ведутся работы по внедрению TerraNet как основы для построения единой информационной среды при создании распределенных тренажерных комплексов для воспроизведения как одиночного, так и группового пилотажа [9]. Такие комплексы, как правило, являются сложным набором разнородных компонентов, подчас разрабатываемых различными производителями (рис. 5). Основными задачами программного комплекса TerraNet здесь является интеграция компонентов и обеспечение их согласованного взаимодействия в реальном времени.

Эксперименты показывают, что текущая версия библиотеки может обслуживать до 70 компонентов РСВР на стандартных ПК в рамках 100 Мбит/с ЛВС при сохранении достаточно высокого уровня согласованности данных и чувствительности равной 1/60 с [7, 10]. В ближайшем будущем планируется увеличить число поддерживаемых компонентов за счет введения поддержки мультисерверной схемы взаимодействия процессов и провести эксперименты по развертыванию РСВР в глобальной сети.

В заключение автор хотел бы выразить признательность д-ру техн. наук, проф. Дзегеленку И. И. за помощь и поддержку при выполнении данной работы.

Работа выполнена при поддержке РФФИ (грант № 11-07-00751-а), Совета по грантам Президента РФ (НШ-7239.2010.9), аналитической ведомственной целевой программы "Развитие научного потенциала высшей школы" (проекты 2.1.2/6718, 2.1.2/13283) и Федеральной целевой программы "Научные и научно-педагогические кадры инновационной России" (Государственный контракт П2227, 16.740.11.0038).

Список литературы

1. **Singhal S., Zyda M.** Networked Virtual Environments: Design and Implementation. USA, MA: ACM Press/Addison Wesley, 1999.

2. **Smed J., Hakonen H.** Algorithms and Networking for Computer Games. UK, Chichester: John Wiley & Sons, 2006.

3. **Cai W., Lee F., Chen L.** An auto-adaptive dead reckoning algorithm for distributed interactive simulation // Proc. of the Thirteenth Workshop on Parallel and Distributed Simulation. 1999. May. P. 82—89.

4. **Strassburger S., Schulze T., Fujimoto R.** Future trends in distributed simulation and distributed virtual environments: results of a peer study // In Proc. of the 40th Conference on Winter Simulation. Miami, Florida. 2008. USA. P. 777—785.

5. **OpenSceneGraph** — an open source high performance 3D graphics toolkit. — Electronic data. — URL: http://www.openscenegraph.org.

6. **Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.** Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб.: Питер, 2007.

7. **Харитонов В. Ю.** Сетевые механизмы обеспечения согласованности данных в распределенных системах виртуальной реальности // Автореферат диссертации на соискание ученой степени кандидата технических наук. — М.: Полиграфический центр МЭИ (ТУ), 2010. 20 с.

8. **Kharitonov V. Y.** A Software Architecture of Distributed Virtual Reality System for Formation Flight Visualization // Proc. of 3rd European Conference for Aero-Space Sciences. Versailles, France. 2009. July 6—9th. — Electronic data. (CD-ROM)

9. **Харитонов В. Ю., Бажин В. А., Рудельсон Л. Е.** Компьютерное воспроизведение виртуальной реальности в современных авиационных тренажерах // Научный Вестник МГТУ ГА. 2011. № 171 (9). С. 158—165.

10. **Kharitonov V. Y.** A Middleware for Integrating Components of Modern Distributed Virtual Reality Systems // Proc. of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2011, Washington, DC, USA. 2011. August 29—31.

УДК 004.413.4

Д. Е. Пальчунов, д-р физ.-мат. наук, доц., вед. науч. сотр., Институт математики СО РАН им. С. Л. Соболева, Новосибирский государственный университет, e-mail: palch@math.nsc.ru,
Г. Э. Яхъяева, канд. физ.-мат. наук, доц., e-mail: gulnara@math.nsc.ru,
А. А. Хамутская, студентка, Новосибирский государственный университет, e-mail: alena.khamutskaya@gmail.com

Программная система управления информационными рисками RiskPanel[1]

Рассмотрены современные подходы к оценке рисков неблагоприятных событий, необходимость в которых возникает при обеспечении информационной безопасности корпоративной информационной системы. Проведен анализ существующих программных систем, предназначенных для оценки таких рисков. Предложен подход к анализу рисков, основанный на построении формальных моделей прецедентов компьютерных атак. Изложены математические основы предлагаемого подхода. Описана программная реализация подхода — программная система RiskPanel.

Ключевые слова: *информационная безопасность, компьютерная атака, управление рисками, корпоративная информационная система, нечеткие модели*

1. Обзор инструментальных средств управления информационными рисками

В настоящее время актуальность эффективного управления рисками в сфере информационной безопасности трудно переоценить [1]. В отчетах крупных компаний часто фигурируют огромные цифры финансовых потерь, понесенных в результате хакерской атаки, утраты ценных сведений и т. п.

Основная задача специалистов, обеспечивающих компьютерную безопасность, — это оперативная реакция на изменения текущего статуса защищенности всех компонентов корпоративной информационной системы. Для этой цели полезно иметь программную систему, позволяющую без необходимости приобретения особых навыков оперативно определить тип атаки, узнать самую свежую информацию о возможных последствиях атак и способах их предотвращения. На настоящее время разработано более сотни различных программных систем управления информационными рисками. Все их можно условно разделить на две группы [2—4]:

- программные системы базового уровня — методики качественного анализа рисков;
- программные системы полного анализа — методики количественного анализа рисков.

Программные системы базового уровня, как правило, используются компаниями, находящимися на уровне 3 зрелости по классификации CMM [5]. К методикам качественного анализа рисков на основе требований ISO 17999 относятся методики COBRA и RA Software Tool.

COBRA. Во второй половине 90-х годов XX века компания C&A Systems Security Ltd. разработала мето-

[1] Работа выполнена при поддержке гранта Федерального агентства по образованию, государственный контракт П-1008, а также междисциплинарного интеграционного проекта фундаментальных исследований СО РАН 119.

дику и инструментарий для анализа и управления информационными рисками под названием COBRA [6]. Эта методика позволяет в автоматизированном режиме выполнить достаточно простой вариант оценивания информационных рисков любой компании. Анализ рисков, выполняемый данным методом, отвечает базовому уровню безопасности. Достоинство методики в ее простоте. Необходимо ответить на несколько десятков вопросов, затем автоматически формируется отчет.

RA Software Tool [7]. Методика базируется на британском стандарте BS 7799 и на методических материалах Британского института стандартов.

К программным системам полного анализа рисков относится инструментарий с более развитыми средствами анализа рисков и управления ими. Такой инструментарий пользуется спросом у организаций, находящихся на уровнях 4 и 5 зрелости по классификации CMM. На четвертом уровне для руководства организации актуальны вопросы измерения параметров, характеризующих режим информационной безопасности. Технология управления режимом информационной безопасности остается прежней, но на этапе анализа рисков применяются количественные методы, позволяющие оценить параметры остаточных рисков, эффективность различных вариантов контрмер при управлении рисками. На пятом уровне ставятся и решаются различные варианты оптимизационных задач в области обеспечения режима информационной безопасности. Рассмотрим самые известные программные системы данного класса.

CRAMM. Метод CRAMM (*CCTA Risk Analysis and Management Method*) [8] был разработан Агентством по компьютерам и телекоммуникациям Великобритании (*Central Computer and Telecommunications Agency*) по заданию Британского правительства и взят на вооружение в качестве государственного стандарта. Он используется начиная с 1985 г. правительственными и коммерческими организациями Великобритании.

Исследование информационной безопасности системы с помощью CRAMM осуществляется в несколько этапов. На первом этапе проводится формализованное описание границ информационной системы, ее основных функций, категорий пользователей, а также персонала, принимающего участие в обследовании. На втором этапе описывается и анализируется все, что касается идентификации и определения ценности ресурсов системы. По завершению данного этапа заказчик имеет качественное описание уровня информационной безопасности своей компании.

Следующий этап позволяет оценить риски либо на основе сделанных оценок угроз и уязвимостей при проведении полного анализа рисков, либо путем использования упрощенных методик для базового уровня безопасности. На последнем этапе проводится поиск адекватных контрмер.

RiskWatch. Методика разработана американской компанией RiskWatch Inc [9]. Используемая в программе методика состоит из четырех этапов. На первом этапе описываются параметры организации: ее тип; состав исследуемой системы; базовые требования в области безопасности. Второй этап — внесение данных, касающихся конкретных характеристик системы. Данные могут вводиться вручную или импортироваться из отчетов, созданных инструментальными средствами исследования уязвимости компьютерных сетей. На этом этапе: описываются ресурсы, потери и классы инцидентов; с помощью опросника, база которого содержит более 600 вопросов, выявляются возможные уязвимости; задается частота возникновения каждой из выделенных угроз, степень уязвимости и ценность ресурсов. Третий этап — оценка рисков нарушения безопасности. Четвертый этап — генерация отчетов.

Digital Security Office 2006 — комплексное решение для управления информационной безопасностью, разработанное российской компанией *Digital Security* [10]. Digital Security Office 2006 включает в себя систему разработки и управления политикой безопасности информационной системы КОНДОР и систему анализа и управления информационными рисками ГРИФ. Система КОНДОР предназначена для обеспечения базового уровня информационной безопасности компании. Система ГРИФ направлена на обеспечение полного анализа информационных рисков. Эта система позволяет проанализировать уязвимости информационной системы и оценить возможный ущерб для компании при реализации потенциальных угроз через найденные уязвимости.

В ходе работы с системой ГРИФ строятся две модели: модель анализа информационных потоков и модель анализа угроз и уязвимостей. Анализ рисков информационной безопасности осуществляется с помощью построения модели информационной системы организации. Для оценки рисков информационной системы организации защищенность каждого ценного ресурса определяется при помощи анализа угроз, действующих на конкретный ресурс, и уязвимостей, через которые данные угрозы могут быть реализованы.

OCTAVE (*Operationally Critical Threat, Asset, and Vulnerability Evaluation*) [11]. Эта методика разработана в университете Карнеги-Мелон, США и предназначена для оценки критичных угроз, активов и уязвимостей.

2. Анализ недостатков существующих программных решений

Первым шагом любой методики управления информационными рисками является идентификация рисков, в том числе их составляющих угроз и уязвимостей. Типовым подходом к решению данной задачи является использование различных стандартных списков классов рисков. Компании, создающие программные системы по информационной безопасности, разрабатывают свои списки рисков (угроз и уязвимостей) информационной безопасности или используют некоторые общепринятые стандарты, либо покупают такие списки у крупных производителей программного обеспечения. В частности, используются каталоги стандартных рисков, угроз и уязвимостей. Примером такого каталога является список *Common Vulnerabilities and Exposures* (CVE) [12] — единый каталог уязвимостей. Трудности реализации такого подхода обусловлены излишней конкретизацией. Заказчик, используя подробные каталоги, может "потонуть" в море рисков, которые система для него идентифицирует.

Вторая трудность, с которой сталкиваются разработчики систем информационной безопасности, это очень динамичное развитие информационных технологий, и, как следствие, непрерывное появление новых угроз и уязвимостей. Разработчики стараются отслеживать появление новых рисков, выпускают новые версии программ. Для решения этой задачи программная система RiskPanel, которая подробно будет описана в разд. 4, имеет модуль полуавтоматического пополнения базы данных прецедентов компьютерных атак, которое осуществляется за счет анализа оперативной информации, представленной в Интернет и других источниках [13—16].

Третья трудность, с которой сталкиваются разработчики современных программных решений, связана с методологией и техникой оценки рисков нарушения информационной безопасности. В каждой программной системе по управлению информационными рисками заложен некий алгоритм качественной или количественной оценки идентифицированных рисков. Как правило, этот алгоритм является "ноу-хау" компании и скрыт для тех, кто не вовлечен в разработку данной программной системы.

Заказчик, работая с предоставленной ему программной системой по информационной безопасности, получает оценку определенного набора рисков и список контрмер, которые необходимо предпринять для ликвидации данных рисков. Однако у него нет возможности восстановить знания о тех прецедентах (конкретных случаях компьютерных атак), на основании которых были получены данные оценки. Теоретические принципы и конкретные математические формулы, на основе которых проводится оценка рисков, являются "ноу-хау" разработчиков программных систем и, соответственно, недоступны пользователю. Как следствие, для пользователя программная система представляет собой "черный ящик", который выдает рекомендации, но аргументация этих рекомендаций, информация, которая стоит за полученными выводами, является недоступной. В результате пользователю, во-первых, сложно оценить достоверность сведений о рисках, и, во-вторых, нет полного понимания, что дальше делать с полученной информацией.

В предлагаемом нами подходе вся исходная информация о прецедентах компьютерных атак хранится в базе знаний системы и, при соответствующем запросе заказчика, становится ему доступной. Более того, при пополнении базы знаний создаются новые классы рисков и система в автоматическом режиме находит необходимые контрмеры для ликвидации этих рисков.

3. Математические основы разрабатываемого подхода

Как было описано выше, существующие программные средства оценки рисков неблагоприятных событий, которые выявляются при обеспечении безопасности информационных систем, имеют определенные недостатки. Один из таких недостатков состоит в том, что при их применении пользователь имеет только несколько чисел, характеризующих риски (например, вероятность данного риска, важность этого риска и т. п.), но остается непонятным, что с этими числами делать дальше. Это, в частности, связано с тем обстоятельством, что содержательная информация, которая представлена за этими числами, недоступна для пользователя. Содержательная информация "спрессована" в числа без возможности ее обратного восстановления по этим числам.

Отмеченный выше недостаток можно устранить, если иметь дело не только с итоговыми числами, представляющими, например, вероятность возникновения неблагоприятных событий и степень их критичности, но и с исходной информацией, из которой эти числа получены — с прецедентами реальных компьютерных атак и других видов нарушения информационной безопасности. Для этого необходимо работать с базой данных прецедентов компьютерных атак. Подробное описание программной системы RiskPanel, позволяющей оценивать риски информационной безопасности исходя из имеющихся преце-

дентов компьютерных атак, будет дано в разд. 4. В настоящем разделе изложим математические основы предлагаемого подхода.

Принципиальное отличие математических основ предлагаемого подхода [17, 18] от стандартных методов оценки рисков и методов актуарной математики [19] состоит в том, что здесь работа ведется не с числовыми оценками возможности срабатывания различных рисков, а с множествами прецедентов, на которых эти риски сработали. Кратко это отличие можно описать следующим образом. При стандартном подходе информация сначала оцифровывается (в другой терминологии фазифицируется), а потом обрабатывается. В рамках предлагаемого подхода вся имеющаяся информация, включающая и описание онтологии предметной области, и эмпирические данные, сначала полностью обрабатывается, и только окончательный результат фазифицируется, превращается в числа из интервала [0, 1]. Такой подход дает возможность на всех шагах обработки информации иметь дело с полностью релевантными данными, не искаженными оцифровкой.

Проиллюстрируем это на примере. Допустим, нужно рассчитать вероятность риска "Неавторизованная модификация информации в базе данных". Предположим, что эта угроза может осуществиться через две различные уязвимости A и B. В методике компании *Digital Security* [10], заказчику предлагается рассчитать вероятности реализации угрозы через каждую из уязвимостей: $P(A)$ и $P(B)$. При помощи формулы $P(A \cup B) = 1 - (1 - P(A))(1 - P(B))$ вычисляется общая вероятность реализации угрозы. Например, если $P(A) = 0{,}5$ и $P(B) = 0{,}5$, то $P(A \cup B) = 0{,}75$. При этом не может быть учтена информация о совместимости или несовместимости событий A и B. Если иметь не только числовые оценки, но и стоящие за ними прецеденты компьютерных атак, то будет известна информация о совместимости данных событий. При этом, если рассматриваемые события несовместимы (т. е. $P(A \cap B) = 0$), то $P(A \cup B) = 1$; если они происходят одновременно, то $P(A \cup B) = 0{,}5$.

В настоящей работе мы рассматриваем конечное множество прецедентов компьютерных атак [1] и, исходя из этих прецедентов, оцениваем вероятности различных утверждений, имеющих отношение к безопасности корпоративной информационной системы. Каждый прецедент формализуется в виде алгебраической системы. Для простоты и удобства рассмотрения, без ограничения общности, можно считать, что у всех алгебраических систем, описывающих прецеденты компьютерных атак, основное множество (с точностью до переобозначения) одно и то же. Также, без ограничения общности, можно считать, что после соответствующего переобозначения основные множества алгебраических систем, описывающих прецеденты, не пересекаются.

Таким образом, каждый прецедент компьютерной атаки описывается алгебраической системой $\mathfrak{U} = \langle A, \sigma \rangle$, где A — основное множество алгебраической системы, а σ — ее сигнатура. Сигнатура σ — это множество понятий, на языке которых описывается данная предметная область: множество различных уязвимостей, угроз, контрмер, последствий и т. п. Будем считать, что у всех прецедентов компьютерных атак сигнатура одна. Обогатим сигнатуру σ, добавив для каждого элемента a константу c_a: обозначим $\sigma_A = \sigma \cup \{c_a \mid a \in A\}$. Алгебраические системы, с помощью которых описываем экземпляры предметной области, принадлежат классу

$$\mathbb{K}(\sigma_A) \rightleftharpoons (\mathfrak{U} = \langle \{ c_a^{\mathfrak{U}} \mid a \in A\}, \sigma_A \rangle | \; c_a^{\mathfrak{U}} \neq c_b^{\mathfrak{U}} \text{ при } a \neq b\}.$$

Как было отмечено выше, считаем, что для разных $\mathfrak{U}, \mathfrak{B} \in \mathbb{K}(\sigma_A)$ выполняется $|\mathfrak{U}| \cap |\mathfrak{B}| = \varnothing$. Через $\wp(X)$ будем обозначать множество всех подмножеств множества X. Через $S(\sigma_A)$ будем обозначать множество всех предложений (замкнутых формул) сигнатуры σ_A.

Алгебраическую систему $\mathfrak{U}$, являющуюся моделью некоторой компьютерной атаки, назовем **прецедентом** этой предметной области. Для каждого набора **прецедентов** E определим прецедентную систему $\mathfrak{U}_E$.

Определение 1. *Пусть $E \subseteq \mathbb{K}(\sigma_A)$ — некоторое множество прецедентов. Прецедентной системой (порожденной множеством E) назовем алгебраическую систему $\mathfrak{U}_E \rightleftharpoons \langle A, \sigma, \tau_E \rangle$, где $\tau_E : S(\sigma_A) \to \wp(E)$, причем для любого предложения φ сигнатуры σ_A выполнено $\tau_E(\varphi) = \{\mathfrak{U} \in E \mid \mathfrak{U} \models \varphi\}$.*

Заметим, что прецедентная система является частным случаем булевозначной модели [18], в которой значениями истинности предложений являются элементы булевой алгебры. В данном случае значениями истинности предложений являются элементы $\to \wp(E)$ — булевой алгебры всех подмножеств множества E.

Рассмотрим множество X всевозможных компьютерных атак (как уже произошедших и известных нам, так и тех, которые еще могут произойти). Очевидно, что в каждый момент времени наше знание об уже произошедших компьютерных атаках конечно. Однако это знание постоянно растет, пополняясь новыми прецедентами. Таким образом, можно предполагать, что потенциально множество X прецедентов компьютерных атак является счетным. При этом достаточно рассматривать только конечные подмножества множества X, как формализацию нашего знания о предметной области в разные моменты времени. Таким

образом, будем рассматривать только конечные множества прецедентов, и, следовательно, только булевозначные модели с конечными булевыми алгебрами. Введем следующее обозначение для класса всех булевозначных моделей с конечными булевыми алгебрами

$$\mathbb{K}^f \rightleftharpoons \{\mathfrak{U}_E \mid E \subseteq \mathbb{K}(\sigma_A) \text{ и } E \text{ — конечно}\}.$$

Основной целью программной системы по управлению информационными рисками является идентификация и оценивание рисков, связанных с информационной безопасностью корпоративной информационной системы. В большинстве методик управления информационными рисками при их оценивании используются объективные и/или субъективные вероятности [3].

Под **объективной вероятностью** понимается относительная частота появления какого-либо события в общем объеме наблюдений или отношение числа благоприятных исходов к общему числу наблюдений. Под **субъективной вероятностью** имеется в виду мера уверенности некоторого эксперта или группы экспертов в том, что данное событие в действительности будет иметь место.

В подходе, представленном в настоящей работе, при идентификации и оценивании рисков также будем использовать объективную и субъективную вероятности. В нашем случае объективная вероятность — это функция истинности $\mu(\varphi)$ в нечеткой модели $\mathfrak{U}_\mu$ (которая будет определена ниже), а субъективная вероятность — это оценка, сделанная экспертом. Опишем сначала методы подсчета объективных вероятностей рисков.

Пусть у нас есть прецедентная модель $\mathfrak{U}_E$, являющаяся математической формализацией базы знаний о прецедентах компьютерных атак. Для того чтобы вычислить объективные вероятности происхождения тех или иных атак, определим понятие фазификации прецедентной модели.

Определение 2. *Модель $\mathfrak{U}_\mu = \langle A, \sigma_A, \mu\rangle$ назовем **фазификацией** прецедентной модели $\mathfrak{U}_E \in \mathbb{K}^f$ (и будем обозначать $\mathfrak{U}_\mu = Fuz(\mathfrak{U}_E)$), если для любого предложения φ сигнатуры σ_A выполнено $\mu(\varphi) = \frac{\|\tau_E(\varphi)\|}{\|E\|}$. Будем обозначать $\mathfrak{U}_\mu \models_\alpha \varphi$, если $\mu(\varphi) = \alpha$.*

Заметим, что полученная модель является нечеткой моделью, так как ее истинностная функция μ принимает свои значения из интервала [0, 1]. Обозначим через $\mathbb{K}^\mu \rightleftharpoons \{\mathfrak{U}_\mu \mid \exists \mathfrak{U}_E \in \mathbb{K}^f : \mathfrak{U}_\mu = Fuz(\mathfrak{U}_E)\}$ класс всех фазификаций прецедентных моделей.

Определение 3. *Предложение φ называется **истинным** на фазификации $\mathfrak{U}_\mu$, если $\mu(\varphi) = 1$.*

Теорема 1. *Предложение φ истинно на любой фазификации $\mathfrak{U}_\mu$ тогда и только тогда, когда оно тождественно истинно в классической логике предикатов.*

На практике, как правило, нет полной информации о рассматриваемой предметной области. Например, никакой эксперт не обладает информацией обо всех компьютерных атаках и нарушениях информационной безопасности, произошедших на настоящий момент времени. Следовательно, нет возможности дать полное описание прецедентной модели, описывающей данную предметную область. По этой причине приходится рассматривать класс $K \subseteq \mathbb{K}^\mu$ прецедентных моделей, описывающих те свойства предметной области, которые уже известны. Для формального описания такой ситуации введем понятие обобщенной нечеткой модели.

Определение 4. *Пусть $K \subseteq \mathbb{K}^\mu$ и $K \neq \varnothing$. Модель $\mathfrak{U}_K = \langle A, \sigma_A, \xi_K\rangle$ назовем **обобщенной нечеткой моделью** (порожденной классом K), если для любого предложения φ сигнатуры σ_A истинностное значение $\xi_K(\varphi)$ является множеством всех объективных вероятностей предложения φ на всех моделях класса K, т. е. выполнено $\xi_K(\varphi) = \{\alpha \in [0,1] \mid \exists \mathfrak{U} \in K : \mathfrak{U} \models_\alpha \varphi\}$.*

Поскольку у нас нет достаточного объема объективной информации, необходимо использовать субъективные вероятности (экспертные оценки) существенных для нас утверждений о предметной области. Пусть U — множество предложений сигнатуры σ_A, про которые известны их субъективные вероятности. Рассмотрим класс $K \subseteq \mathbb{K}^\mu$ таких моделей $\mathfrak{U}_\mu$, у которых для каждого предложения φ из U значение $\mu(\varphi)$ совпадает со значением субъективной вероятности истинности этого предложения φ. Этот факт означает, что на всех предложениях из U значения субъективной и объективной вероятностей совпадают. Для такого класса K верна следующая теорема.

Теорема 2. *На обобщенной нечеткой модели $\mathfrak{U}_K = \langle A, \sigma_A, \xi_K\rangle$ для любого предложения φ сигнатуры σ_A истинностное значение $\xi_K(\varphi)$ является интервалом рациональных чисел.*

Заметим, что кроме совпадения значений $\mu(\varphi)$ на классе K можно также потребовать истинность онтологии предметной области. Однако и при этом требовании Теорема 2 останется верной: можно считать, что все предложения, входящие в онтологию, входят и во множество U, при этом их субъективная вероятность равна 1. Более подробную информацию по обобщенным нечетким моделям можно найти в работах [17, 18].

4. Описание системы RiskPanel

Для того чтобы эффективно управлять рисками при обеспечении информационной безопасности подконтрольных систем, необходимо иметь средства анализа уровня их защищенности, новых видов угроз информационной безопасности и способов их реализации. Представленные на рынке продукты риск-менеджмента в области информационной безопасности основаны на том, что анализ рисков нарушения безопасности ресурсов информационных систем организации проводится разово или с низкой периодичностью, например, раз в полгода. При этом проводится лишь довольно общий анализ без учета того, что ситуация в компании может меняться значительно чаще. Поскольку сфера информационных технологий очень динамична, необходимо проводить анализ рисков постоянно, учитывая происходящие в мире прецеденты нарушения информационной безопасности.

Следующий вопрос — что конкретно пользователь должен делать с информацией о рисках, которую ему выдала программная система? Здесь можно выделить следующие три момента и, соответственно, вытекающие из них задачи.

- Во-первых, пользователю необходимо понять, насколько оценка тех или иных рисков соответствует текущей конфигурации оборудования и программного обеспечения компании.
- Во-вторых, нужно выяснить, насколько актуальна информация о различных рисках. Информация об одних из них могла быть получена совсем недавно, в то время как оценка вероятности и важности других рисков может быть основана на достаточно старой информации.
- В-третьих, необходимо понять, какие выводы нужно сделать из имеющейся информации о рисках: какие действия нужно предпринять заранее, как идентифицировать факт срабатывания риска, как устранить последствия компьютерной атаки.

Программная система RiskPanel направлена на эффективное решение перечисленных задач. В отличие от конкурентных программных решений, она нацелена на постоянный мониторинг угроз безопасности корпоративной информационной системы. Анализ и учет выявленных рисков, планирование действий по недопущению компьютерных атак и ликвидации их последствий происходит с учетом опыта уже произошедших в мире нарушений безопасности и понесенных потерь. Отличительной чертой системы является то обстоятельство, что связь между оценкой конкретных рисков и информацией о произошедших компьютерных атаках является для пользователя прозрачной и понятной.

Анализ рисков в системе RiskPanel заключается в том, что программный комплекс перманентно оценивает ситуацию информационной безопасности в компании, показывает администратору самые опасные риски нарушения безопасности, причины, вследствие которых они проявляются, возможные методы их предотвращения. Все эти действия проводятся с учетом имеющейся и постоянно обновляющейся базы прецедентов компьютерных атак. Пользователю представляется список конкретных рисков, которые угрожают его компании и предлагаются возможные контрмеры, которые помогут закрыть уязвимости системы. Риски рассчитываются на основе знаний о прецедентах компьютерных атак и конфигурации информационной системы компании. Конфигурация такой системы представляет собой оборудование, программное обеспечение, информационные потоки, персонал, финансовые потоки, компьютерные сети и др.

Программный комплекс RiskPanel имеет модульную структуру, позволяющую в дальнейшем подключать к нему новые модули. Например, к основным модулям была подключена система визуализации рисков, основанная на применении ДСМ-метода [20] и анализа формальных понятий [21]. В настоящее время программный комплекс имеет перечисленные далее три основных модуля.

1-й модуль. База знаний о прецедентах компьютерных атак и программная система постоянного оперативного пополнения базы данных. Задача этого модуля осуществлять:

— пополнение информации по компьютерному терроризму и информационной безопасности;

— оперативное реагирование на появление новых видов компьютерных атак;

— предоставление интерфейса для ручной правки и внесения новой информации по компьютерной безопасности.

Данный модуль реализован на платформе OntoBox [22]. Эта платформа содержит систему хранения и обработки знаний, основанную на онтологиях. В основе OntoBox лежит идея использования "интеллектуальных" средств математической логики для решения задач построения информационных ресурсов. Система OntoBox позволяет применять логику описаний (*Description Logic*) [23] для представления данных. Это дает возможность осуществлять логический вывод в целях получения новых знаний, исходя из имеющихся данных о прецедентах компьютерных атак. Информация о прецедентах компьютерных атак

постоянно пополняется из Интернет и других источников, обеспечивая, тем самым, актуальный анализ ситуации. Пополнение происходит в полуавтоматическом режиме.

2-й модуль. Программная система оценки рисков компьютерной атаки и уязвимостей корпоративной информационной системы. Задача этого модуля осуществлять:

— риск-менеджмент безопасности корпоративной информационной системы;

— превентивные меры по предотвращению деструктивных воздействий, способных нанести ущерб компьютерным системам и сетям;

— пресечение компьютерной атаки на начальном этапе ее реализации;

— оперативное устранение последствий компьютерной атаки и минимизацию причиненного ущерба.

3-й модуль. Точная настройка системы компьютерной безопасности RiskPanel на конкретную корпоративную информационную систему. Задачи этого модуля:

- создание профиля данной корпоративной информационной системы, включающего основные параметры оборудования и программного обеспечения;
- организация выбора из базы данных информации по компьютерной безопасности, актуальной именно для данной корпоративной информационной системы;
- пополнение базы данных, исходя из профиля данной корпоративной информационной системы.

Программная система RiskPanel позволяет администратору информационной безопасности проводить достаточно тонкий анализ прецедентов компьютерных атак, имеющихся в базе знаний. Благодаря тому, что система хранения и обработки знаний OntoBox основана на логике описаний, возможна отработка (выяснение истинности) любых утверждений, выразимых в логике описаний, о прецедентах компьютерных атак, их связях с угрозами, уязвимостями системы информационной безопасности, симптомами начала компьютерной атаки, ее последствиями, возможными контрмерами и др. Заметим, что класс утверждений, которые могут быть выражены в логике описаний, является достаточно широким: логика описаний служит одним из средств онтологического моделирования предметных областей. Например, логика описаний лежит в основе языка описания онтологий OWL, продвигаемого WWW-консорциумом в рамках проекта Semantic Web.

Система визуализации позволяет пользователю удобным для него способом отслеживать связи между прецедентами компьютерных атак, угрозами, уязвимостями, симптомами, последствиями и контрмерами. Более того, пользователь может оценить напрямую связи между уязвимостями и симптомами, уязвимостями и контрмерами или последствиями, симптомами и последствиями и т. д. — в любых сочетаниях.

Заключение

В работе рассмотрены современные подходы к оценке рисков реализации компьютерных атак, необходимость в которых возникает при обеспечении безопасности корпоративной информационной системы. Проведен анализ достоинств и недостатков существующих программных систем, предназначенных для оценки таких рисков. Одним из недостатков традиционно используемых систем оценки рисков является тот факт, что пользователю предоставляется только числовая информация по оценке рисков, но не показывается, каким образом и на основе каких данных эти числовые оценки рисков были получены. В работе предложен подход к анализу рисков, направленный на устранение этого недостатка. Данный подход основан на анализе прецедентов реально произошедших компьютерных атак. Прецеденты компьютерных атак представляются в виде, явно доступном для пользователя. С каждым прецедентом связываются уязвимости системы информационной безопасности, вследствие которых этот прецедент может произойти, симптомы начала компьютерной атаки, ее последствия, контрмеры и др.

Изложены математические основы предлагаемого подхода к анализу прецедентов компьютерных атак и оценки рисков информационной безопасности. Исследуется специальный класс алгебраических систем — прецедентные модели, являющиеся частным случаем булевозначных моделей с атомной (в частности, с конечной) булевой алгеброй. Для получения числовой информации по оценке рисков информационной безопасности проводится фазификация булевозначных моделей: по прецедентной модели строится нечеткая модель, в которой каждому предложению сопоставлена вероятность его истинности.

Описана программная реализация подхода к оценке рисков информационной безопасности, основанного на анализе прецедентов компьютерных атак — программная система RiskPanel. Система RiskPanel позволяет администратору информационной безопасности: в явном виде работать с прецедентами компьютерных атак, осуществляя превентивные меры, направленные на их предотвращение и недопущение нанесения ущерба компьютерным системам и сетям; пресекать деструктивные воздействия на начальных

этапах их реализации; оперативно устранять последствия таких воздействий для минимизации ущерба, причиненного компании.

Список литературы

1. **Васенин В. А.** Информационная безопасность и компьютерный терроризм // Научные и методологические проблемы информационной безопасности / Под ред. В. П. Шерстюка. М.: МЦМО, 2004 г.

2. **Варфоломеев А. А.** Управление информационными рисками: учеб. пособие. М.: РУДН, 2008. 158 с.

3. **Петренко С. А., Симонов С. В.** Управление информационными рисками. Экономически оправданная информация. М.: ДМК Пресс, 2005. 384 с.

4. **Нестеров С. А.** Анализ и управление рисками в информационных системах на базе операционных систем Microsoft. Библиотека учебных курсов MSDN Academic Alliance. 2007.

5. **Paulk M. C., Curtis B., Chrissis M. B., Weber C. V.** Capability Maturity Model for Software, version 1.1 // CMU/SEI-93-TR-024. 1993. February.

6. **ПО "COBRA".** URL: http://www.pcorp.u-net.com/risk.htm Интернет-портал компании-разработчика C & A Systems Security Ltd.

7. **RA Software Tool.** URL: http://www.aexis.de/RA%20 ToolPage.htm Демонстрационная версия метода.

8. **ПО "CRAMM".** URL: http://www.cramm.com/ Интернет-портал компании-разработчика.

9. **ПО "RiskWatch".** URL: http://www.riskwatch.com/ Интернет-портал компании-разработчика Risk Watch International.

10. **ПО "Digital Security Office 2006".** URL: http://dsec.ru/ Интернет-портал компании-разработчика Digital Security.

11. **ПО "OCTAVE".** URL: http://www.cert.org/octave/ Интернет-портал компании-разработчика CERT® Coordination Center.

12. **Каталог уязвимостей** CVE, URL: http://cve.mitre.org/cve.

13. **Пальчунов Д. Е.** Решение задач поиска информации на основе онтологий // Бизнес-информатика. 2008. Т. 1. С. 3—13.

14. **Пальчунов Д. Е.** Поиск и извлечение знаний: порождение новых знаний на основе анализа текстов естественного языка // Философия науки. 2009. № 4 (43). С. 70—90.

15. **Pal'chunov D. E.** Virtual catalog: the ontology-based technology for information retrieval // Lecture Notes in Artificial Intelligence, 6581, Springer-Verlag Berlin Heidelberg. 2011. P. 164—183.

16. **Власов Д. Ю., Пальчунов Д. Е., Степанов П. А.** Автоматизация извлечения отношений между понятиями из текстов естественного языка // Вестник НГУ. Серия: Информационные технологии. 2010. Т. 8, Вып. 3. С. 23—33.

17. **Palchunov D. E., Yakhyaeva G. E.** Interval fuzzy algebraic systems // Proc. of the Asian Logic Conference 2005. World Scientific Publishers. 2006. P. 23—37.

18. **Пальчунов Д. Е., Яхъяева Г. Э.** Нечеткие алгебраические системы // Вестник НГУ. Серия: Математика, механика, информатика. 2010. Т. 10. Вып. 3. С. 75—92.

19. **Promislow S. D.** Fundamentals of Actuarial Mathematics. John Wiley and Sons, 2011. 466 p.

20. **ДСМ-метод** автоматического порождения гипотез: Логические и эпистемологические основания / Сост. Аншаков О. М., Фабрикантова Е. Ф.; Под ред. О. М. Аншакова. М.: Книжный дом "ЛИБРОКОМ", 2009. 432 с.

21. **Ganter B., Stumme G., Wille R.** eds. Formal Concept Analysis: Foundations and Applications // Lecture Notes in Artificial Intelligence. 2005. N 3626. 372 p.

22. **Малых А. А., Манцивода А. В.** Онтобокс: онтологии для объектов // Известия Иркутского государственного университета. 2009. Т. 2. № 2. С. 94—104.

23. **Baader F.** The Description Logic Handbook. New York: Cambridge University Press. 2003. 555 p.

ИНФОРМАЦИЯ

7 февраля 2012 г. в здании Правительства Москвы (ул. Новый Арбат, 36) состоится центральное в Российской Федерации отраслевое мероприятие — 14-й Национальный форум информационной безопасности Инфофорум (Инфофорум-2012).

Более тысячи специалистов соберутся для подведения итогов 2011 г. и определения основных задач на 2012 г. — год решения важнейших проблем в процессе создания информационного общества в Российской Федерации: внедрения универсальных электронных карт, систем межведомственного электронного взаимодействия, перевода государственных услуг в электронную форму. Узнайте на Инфофоруме-2012:

- о предложенном Российской Федерацией проекте концепции международной конвенции "Об обеспечении международной информационной безопасности" и других стратегиях международной, национальной и корпоративной безопасности;
- о механизмах защиты информации при использовании универсальной электронной карты;
- о построении унифицированной системы обеспечения информационной безопасности в госорганах и при реализации приоритетных проектов.

Связаться с Оргкомитетом можно по тел./факсу: **+7 (495) 609-67-85; e-mail: info@infoforum.ru.**

Контактное лицо — **Ульянова Мария.**

Сайт Инфофорума-2012 — **2012.infoforum.ru**

УДК 621.391:004.31:519.85

А. В. Чичагов, науч. сотр., Вычислительный Центр им. А. А. Дородницына РАН,
e-mail: mail2tchi@yandex.ru

Оценка адекватности класса спецпроцессоров цифровой обработки сигналов*

Предлагается критерий оценки адекватности класса спецпроцессоров цифровой обработки сигналов. Приводятся описания вычислительной модели, сценария и результатов серии вычислительных экспериментов по оценке адекватности специализированных вычислителей дискретного преобразования Гильберта.

Ключевые слова: *цифровая обработка сигналов, дискретное преобразование Гильберта, вычислительная модель, вычислительный эксперимент*

Как известно, математическая модель представляет приближенное описание какого-либо класса явлений, выраженное с помощью математической символики [1] и, добавим, с интерпретацией примитивов и результатов моделирования на языке понятном специалистам соответствующей предметной области. Указанное добавление фактически подразумевает наличие сообщества специалистов и института экспертизы, т. е. возможность проверки "научности" или достоверности модели, а также оценку качества предполагаемых моделей. Отметим, что в условиях инновационного пути развития экономики, роль института экспертизы существенно повышается.

Вычислительная модель представляет некоторую аппроксимацию определенного математического преобразования класса физических процессов или сигналов. Очевидно, существует множество возможностей для выбора конструктивной аппроксимации (алгоритма) математического преобразования сигнала. Специализированный вычислитель или спецпроцессор, который реализует заданный алгоритм, в зависимости от выбранной формы реализации и аппаратно-программной среды исполнения может представлять устройство, программу или процедуру.

Специализированный вычислитель цифровой обработки сигнала (ЦОС) обычно содержит набор конфигурационных параметров, значения которых требуется задать при его использовании. Таким образом, пользователь путем выбора, как самого (типа) вычислителя, так и значений его параметров может в определенной степени управлять "качеством" результата или точностью цифровой обработки сигнала.

В данной работе основное внимание акцентируется на оценке качества, а точней адекватности пары "алгоритм-программа" математической модели в триаде "математическая модель—алгоритм—программа", которая была сформулирована академиком А. А. Самарским [2]. При этом адекватность исходной математической модели представляемому классу явлений/процессов в данной статье не обсуждается.

Критерий оценки адекватности спецпроцессоров ЦОС

Понятие "качества" как степень "соответствия идеалу" можно рассматривать в иной шкале оценок как "несоответствие или отклонение от идеала", т. е. как недостатки или дефекты системы или, в нашем случае выходного сигнала. В детерминированных системах связь между системой и сигналом однозначная, поэтому если сместить акцент рассмотрения с результата цифровой обработки сигнала на систему ЦОС, то имеющиеся недостатки или дефекты ЦОС можно классифицировать как:

- **логические и арифметические дефекты** (ошибки функционирования ПО, погрешности цифрового представления, например, ошибки квантования, переполнение и др.);

* Работа выполнена при финансовой поддержке РФФИ, проект № 11-01-00900.

- **алгоритмические и методические дефекты** (грубость или неадекватность используемых алгоритмов, несоответствие текущему заданию выбранных пользователем терминальных значений параметров и др.).

Таким образом, имеется, по крайней мере, два вида дефектов, с которыми могут столкнуться пользователи и которые требуется обнаружить и исправить. Существуют и достаточно хорошо известны различные средства тестирования и отладки аппаратно-программного обеспечения для эффективного обнаружения и исправления дефектов первого вида, поэтому в данной работе они не рассматриваются. В настоящей работе рассматривается способ обнаружения дефектов цифровой обработки сигналов второго вида, или, иначе, критерий оценки адекватности (т. е. степени соответствия идеалу) специализированных вычислителей ЦОС.

Алгоритмическим дефектом назовем границу точности или погрешности преобразования сигнала вычислителем, реализующим заданный алгоритм математического преобразования для заданного класса сигналов при заданных значениях вектора параметров вычислителя.

Методическим дефектом будем называть выбор пользователем некорректных или нерациональных значений вектора параметров вычислителя, при которых погрешность результата цифровой обработки сигнала соответственно выше (т. е. слишком грубая обработка данных) или значительно ниже (т. е. очень точная обработка данных, обычно слишком дорогая и технико-экономически не обоснованная) требуемой (заданной) границы точности цифровой обработки сигнала.

Очевидно, что если пользователю известны границы точности вычислителя при разных значениях вектора параметров, то выбор корректных и рациональных значений вектора параметров не представляет особого труда. Таким образом, методические дефекты вызваны отсутствием у пользователя опыта или необходимой информации, т. е. являются "субъективными" и, следовательно, теоретически их можно избежать. Именно поэтому важно предоставить пользователям конкретного спецпроцессора необходимую информацию, которая позволит избежать, по крайней мере, ряда методических дефектов при его практическом использовании.

При разработке тестов баз данных, информационных систем, офисных или корпоративных приложений часто используется 1-битовая шкала оценок (зачет/незачет). Однако для оценки адекватности спецпроцессоров ЦОС использование такой простой шкалы обычно не является достаточно информативным. Это связано с тем, что величина погрешности результата преобразования ЦОС является "гладкой функцией" вектора параметров вычислителя. На практике для оценки погрешности цифровой обработки сигнала обычно используют метод вычисления среднего квадратического отклонения (СКО). При этом часто ограничиваются общей оценкой погрешности ЦОС, что иногда может оказаться недостаточно. Ниже описывается критерий оценки адекватности специализированных вычислителей цифровых сигналов на конкретном примере вычислителя дискретного преобразования Гильберта (ДПГ).

Выберем семейство комплексных гармоник $z_s = A_0\exp(2\pi j F_s t)$, где A_0 — амплитуда, F_s — частота гармоники, t — время и $j = \sqrt{-1}$, в качестве семейства тестовых цифровых сигналов. Для рассматриваемого семейства функций оценка мощности/дисперсии гармоники $\overline{\sigma_s^2} = \langle |z_s(t)|^2 \rangle \equiv A_0^2$ является величиной явно независимой как от частоты гармоники, так и от длины интервала оценивания, что достаточно удобно для вычисления оценок.

Набор исходных (*source*) тестовых выборок данных или, иначе, набор цифровых сигналов источника $\left\{ \mathbf{z}_s^{(src)} \right\}$ сформируем в виде набора пар временных рядов:

$$x_s^{(src)}(i) = A_0\cos(2\pi(s/2L_0)i),$$
$$y_s^{(src)}(i) = A_0\sin(2\pi(s/2L_0)i),$$

где $i = [F_D^{(src)}t]$ — индекс (смещение по времени) элемента выборки данных ($0 \leqslant i < N_0$); $s/2L_0 \sim F_s/F_D^{(src)}$ — нормализованная частота гармоники; s — спектральный индекс (смещение по частоте) гармоники сигнала источника; $F_D^{(src)}$ — частота дискретизации сигнала источника; N_0 — объем тестовой выборки данных ("длина" фрагмента гармоники); L_0 — размерность набора (спектра) гармоники. По условию Найквиста $F_s/F_D^{(src)} < 1/2$, поэтому полагаем, $s \in 0, 1, ..., L_0 - 1$. Очевидно, что элементы набора исходных тестовых цифровых сигналов имеют одинаковую мощность/дисперсию, равную $\sigma_s^2 = \|\mathbf{z}_s^{(src)}\|^2 \equiv A_0^2$ для всех $s \in 0, 1, ..., L_0 - 1$.

Набор ожидаемых (*expected*) тестовых выборок данных $\{\mathbf{z}_s^{(exp)}\}$, соответствующий исследуемому преобразованию, в данном случае, преобразованию Гильберта [3], набора исходных тестовых цифровых сигналов, сформируем в виде набора пар временных рядов:

$$x_s^{(exp)}(n) = -(1 - \delta(s))A_0\cos(2\pi K_D^{-1}(s/2L_0)n),$$

$$y_s^{(exp)}(n) = -(1 - \delta(s))A_0\sin(2\pi K_D^{-1}(s/2L_0)n),$$

где n — целочисленный сдвиг (смещение по времени) элемента выборки данных, $K_D = F_D^{(dst)}/F_D^{(src)}$ — заданный коэффициент редискретизации, $F_D^{(dst)}$ — частота дискретизации преобразованного сигнала, $\delta(s)$ — символ Кронекера. Элементы набора ожидаемых тестовых цифровых сигналов также имеют одинаковую мощность/дисперсию, равную $\sigma_s^2 = \|\mathbf{z}_s^{(exp)}\| \equiv A_0^2$ для всех $s \in 1, ..., L_0 - 1$ кроме случая $s = 0$, для которого, очевидно, $\sigma_{s=0}^2 = \|\mathbf{z}_{s=0}^{(exp)}\|^2 \equiv 0$.

Набор фактических (*actual*) тестовых цифровых сигналов $\{\mathbf{z}_s^{(act)}\}$, представляющих результат цифровой обработки набора исходных тестовых цифровых сигналов $\{\mathbf{z}_s^{(src)}\}$, процедурой цифровой обработки сигналов **DSP** сформируем в виде:

$$\mathbf{x}_s^{(act)} = \mathbf{DSP}(\mathbf{p}, \mathbf{x}_s^{(src)}),\ \mathbf{y}_s^{(act)} = \mathbf{DSP}(\mathbf{p}, \mathbf{y}_s^{(src)}),$$

где **DSP** (*Digital Signal Processor*) — исследуемая процедура цифровой обработки сигналов, в нашем случае процедура ДПГ, **p** — вектор параметров процедуры.

Для оценки адекватности исследуемого цифрового преобразования сигнала определим оценку частотной кривой адекватности, которая строится на основе фазосогласованных или минимальных средних квадратических отклонений (МСКО):

$$\overline{\varepsilon_s^2} = \min_{\Delta\varphi} \|\mathbf{z}_s^{(act)} - e^{-j\Delta\varphi}\mathbf{z}_s^{(exp)}\|^2,$$

где s — спектральный индекс (номер гармоники), $\Delta\varphi$ — возможный фазовый сдвиг между гармониками, соответствующими ожидаемому и фактическому тестовым сигналам, который может быть обусловлен реализацией цифрового преобразования.

Оптимальное значение фазового сдвига легко найти:

$$\Delta\varphi_s = -j\ln\sqrt{\langle \mathbf{z}_s^{(act)} | \mathbf{z}_s^{(exp)} \rangle / \langle \mathbf{z}_s^{(exp)} | \mathbf{z}_s^{(act)} \rangle},$$

где $\langle \mathbf{z}_1 | \mathbf{z}_2 \rangle = \frac{1}{(N_2 - N_1)} \sum_{n=N_1}^{N_2-1} z_1(n) z_2^*(n)$ — определение скалярного произведения комплексных векторов; N_1, N_2 — соответственно начальная и конечная границы фрагментов цифровых сигналов, по которым оценивается МСКО; * обозначает операцию комплексного сопряжения. При этом границы фрагментов выбирают таким образом, чтобы исключить влияние краевых эффектов вычислительной процедуры преобразования цифрового сигнала на значение МСКО.

Подставив данное значение фазового сдвига в приведенное выше выражение для набора величин МСКО, получим:

$$\overline{\varepsilon_s^2} = \|\mathbf{z}_s^{(act)}\|^2 + \|\mathbf{z}_s^{(exp)}\|^2 - 2\sqrt{\langle \mathbf{z}_s^{(act)} | \mathbf{z}_s^{(exp)} \rangle \langle \mathbf{z}_s^{(exp)} | \mathbf{z}_s^{(act)} \rangle}.$$

Чтобы можно было легко сравнивать наборы МСКО, соответствующие различным вычислительным процедурам или разным значениям параметров вычислительных процедур цифровой обработки сигналов, необходимо определить унифицированную шкалу оценок. В качестве такой шкалы удобно использовать шкалу децибел и представить набор МСКО или оценку частотной кривой адекватности (ЧКА) вычислительной модели ЦОС в виде:

$$\zeta_s^2 = 10\lg\left(\overline{\varepsilon_s^2}/\sigma_s^2\right)\ [\text{Дб}],$$

где в качестве нормировочного коэффициента выбрана мощность сигнала источника.

Предложенная оценка ЧКА представляет набор значений ("кривую") нормализованных спектральных погрешностей, которая соответствует тройке информационных объектов, описывающих вычислительную модель цифрового преобразования сигнала, а именно

- **имени** преобразования (естественно-языковому идентификатору, который требуется для использования в информационном "мире" или библиографическом пространстве);
- **декларации** преобразования (определению ожидаемых результатов преобразования для класса сигналов);
- **реализации** преобразования (алгоритму и конструкции специализированного вычислителя, выполняющего декларированное преобразование цифровых сигналов "более или менее удовлетворительно").

Как показывает практика, оценка ЧКА является достаточно информативным метрологическим показателем качества или, более точно, адекватности вычислительной модели (реализации) определению (декларации) преобразования сигнала.

Алгоритм дискретного преобразования Гильберта

Пусть $\mathbf{u} = u\left[i\,T_D^{(s)}\right]$ и $\mathbf{v} = v\left[n\,T_D^{(d)}\right]$ — входная и выходная выборки цифровых сигналов, связанные как априори предполагается, преобразованием Гильберта [3], а $T_D^{(s)} = 1/F_D^{(s)}$, $T_D^{(d)} = 1/F_D^{(d)}$ — шаг дискретизации соответственно входной (*source*) и

выходной (*destination*) выборок цифровых сигналов. Дискретное преобразование Гильберта с редискретизацией цифрового сигнала формально можно записать в виде:

$$v\left[n\,T_D^{(d)}\right] = \sum_{i=-\infty}^{\infty} g\left(n\,T_D^{(d)} - i\,T_D^{(s)}\right) u\left[i\,T_D^{(s)}\right],$$

где $g\left(F_D^{(s)}\tau\right) = \dfrac{1-\cos\pi F_D^{(s)}\tau}{\pi F_D^{(s)}\tau}$ — весовая функция ядра или относительный вклад элемента $\mathbf{u} = u\left[i\,T_D^{(s)}\right]$ входной выборки в текущее значение элемента $\mathbf{v} = v\left[n\,T_D^{(d)}\right]$ выходной выборки является асимптотически убывающей функцией модуля аргумента.

Определим индекс $\tilde{i}[n]$ элемента входной выборки, который наиболее близко "по физическому времени" соответствует индексу n элемента выходной выборки

$$\tilde{i}[n] = \left(n F_D^{(s)}\right) / F_D^{(d)}$$

и представим аппроксимацию ДПГ в виде:

$$v\left[n\,T_D^{(d)}\right] =$$

$$= \sum_{m=-M}^{M} g'\left(n\,T_D^{(d)} - (\tilde{i}[n]+m)\,T_D^{(s)}\right) u\left[(\tilde{i}[n]+m)\,T_D^{(s)}\right],$$

где $m = i - \tilde{i}[n]$ — относительный индекс элементов входной выборки или смещение относительно центра окна/сегмента аппроксимации, $M = M_a/2$ — полуширина сегмента, M_a — ширина или апертура окна аппроксимации ДПГ, $g'\left(F_D^{(s)}\tau\right) = g(F_D^{(s)}\tau)\cos^{\beta}\left(\dfrac{\pi F_D^{(s)}\tau}{2M+1}\right)$ — сглаженная весовая функция ядра, $\cos^{\beta}(...)$ — эмпирическая формирующая функция окна аппроксимации, $\beta = 0, 1, 2$ — параметр "гладкости" окна.

Введем коэффициент редискретизации цифрового сигнала $K_D = F_D^{(dst)} / F_D^{(src)}$ и перепишем приведенную выше аппроксимацию ДПГ в виде простого "академического" алгоритма:

ВХОД: $\mathbf{p} = (M_a, K_D, \beta)$, $\mathbf{u} = \{u[i]\}$

Инициализация: $n := 0$, $i := 0$;

пока ($i < \boldsymbol{u}.len$) повторять

$i := [n/K_D]$;

$$v[n] := \sum_{m=-M}^{M} u[i+m]\,g'(n/K_D - i - m);$$

$n := n + 1$;

ВЫХОД : $\mathbf{v} = \{v[n]\}$.

Рассмотренный "академический" алгоритм ДПГ предполагает значительные вычислительные затраты, так как при вычислении каждого текущего значения выходной выборки цифрового сигнала требуется пересчет "ядра" преобразования. Этих вычислений, однако, можно избежать, если модифицировать структуру рассмотренного алгоритма.

Действительно, коэффициент редискретизации цифрового сигнала K_D можно аппроксимировать рациональным числом или дробью $K_D = L/N$, где числитель L можно интерпретировать как коэффициент интерполяции, а знаменатель N — как коэффициент децимации.

В этом случае множество возможных значений минимальных временных сдвигов $\tau = n\,T_D^{(d)} - i\,T_D^{(s)}$ между ближайшими текущими элементами входной и выходной выборок "вырождается" в конечный набор значений, число элементов которого определяется значением L.

Из этого следует практическая реализуемость вычисления полного набора возможных значений весовой функции ядра для заданного значения коэффициента редискретизации. Вычисленные значения весовой функции затем можно использовать в процессе цифровой обработки сигнала.

Иными словами, рассмотренный выше "академический" алгоритм ДПГ можно представить в виде "автокоммутируемого" набора линейных цифровых фильтров с постоянными коэффициентами или, иначе, полифазного фильтра. Таким образом "практический" алгоритм ДПГ состоит из двух процедур [4]:

- процедуры проектирования (вычисления коэффициентов) набора линейных цифровых фильтров;
- процедуры полифазной линейной цифровой фильтрации данных.

Процедура вычисления коэффициентов набора линейных цифровых фильтров легко модифицируется. Это позволяет, в частности, использовать формирующие функции окна аппроксимации, отличные от рассмотренных выше. Заметим также, что при отладке и тестировании программ, реализующих "практический" алгоритм ДПГ, можно использовать результаты цифровой обработки данных программой, реализующей "академический" алгоритм ДПГ.

Апробация вычислительной модели преобразования сигнала

Апробация вычислительной модели представляет собой серию вычислительных экспериментов, в которых исследовалось поведение оценки ЧКА от значений апертуры и параметра "гладкости" окна аппроксимации. Для вычисления оценок ЧКА использовалось контрольно-измерительное приложение [5], а для построения графиков оценок ЧКА — пакет OpenOffice.org.

Сценарий вычислительного эксперимента можно представить в виде следующей последовательности действий.

1. Определить значения параметров сессии вычислительного эксперимента, т. е.
   - значения параметров процедуры синтеза тестовых сигналов;
   - значения параметров процедуры цифровой обработки сигнала;
   - значения параметров процедуры представления информационного отчета.
2. Вычислить набор исходных (*sources*) тестовых цифровых сигналов.
3. Вычислить набор ожидаемых (*expected*) тестовых цифровых сигналов.
4. Вычислить набор фактических (*actual*) тестовых цифровых сигналов.

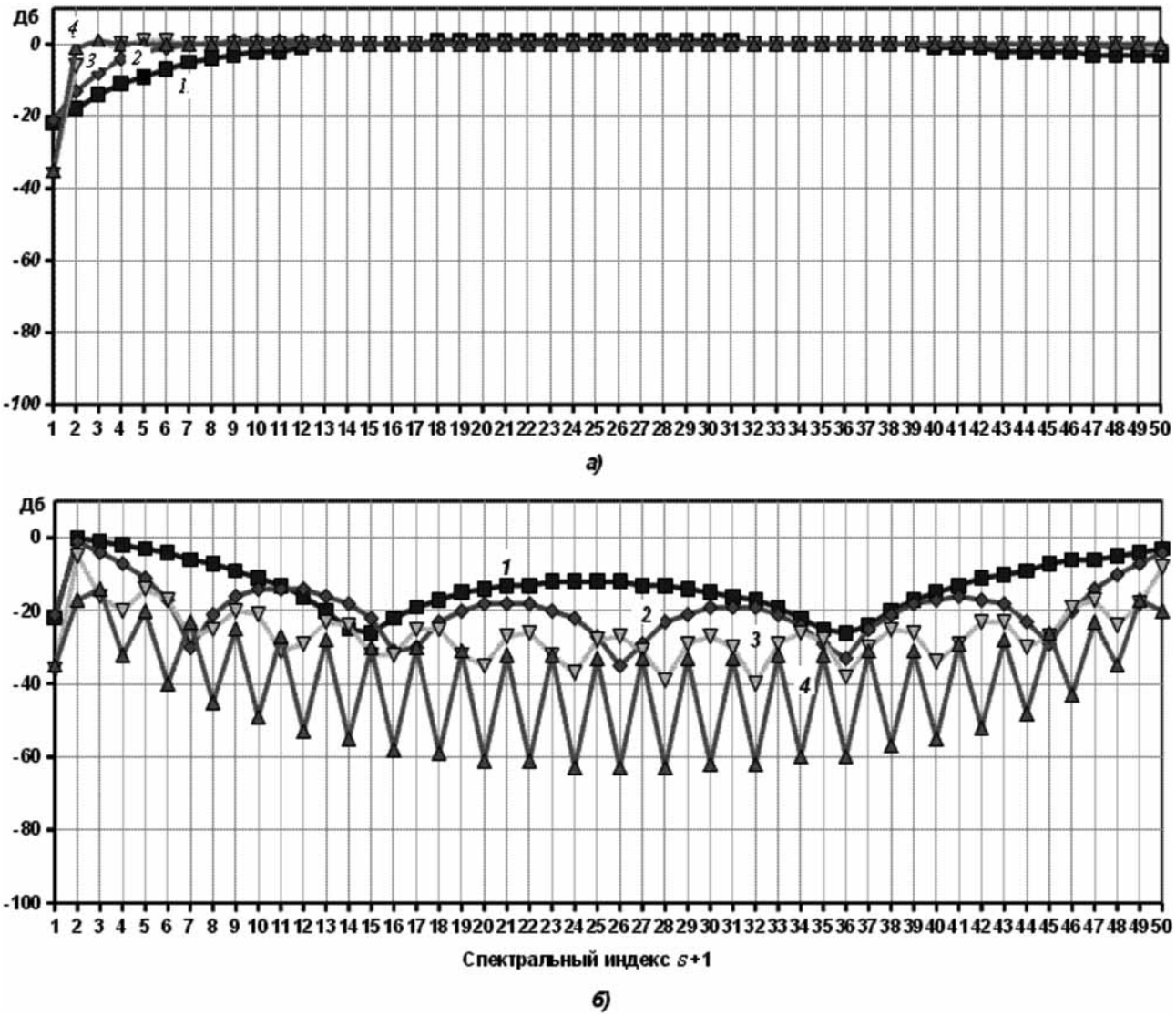


Рис. 1. Графики оценок АЧХ (*а*) и ЧКА (*б*) модели ДПГ (β = 0) при различных значениях апертуры.
Кривые *1—4* соответствуют апертуре M_a = 5, 10, 25, 50 при K_D = 33,3, L_0 = 50, A_0 = 10 000

5. Вычислить оценку ЧКА, соответствующую заданным значениям параметров.

6. Скомпилировать информационный отчет с результатами текущей сессии вычислительного эксперимента для последующего анализа и обсуждения.

Информационный отчет с результатами вычислительного эксперимента должен содержать информацию о сессии вычислительного эксперимента, в частности, следующее:

- значения параметров процедуры синтеза тестовых сигналов $\mathbf{q} = (A_0, N_0, L_0)$;
- значения параметров дискретного преобразования Гильберта $\mathbf{p} = (M_a, K_D, \beta)$;
- оценку ЧКА $\overline{\zeta_s^2}(\mathbf{q}, \mathbf{p})$, соответствующую указанным значениям параметров процедуры синтеза тестовых сигналов и процедуры ДПГ, где A_0 — значение амплитуды сигнала источника (амплитуда гармоники), N_0 — объем тестовой выборки данных (длина фрагмента сигнала источника "***u***.*len*"), L_0 — размерность набора тестовых выборок данных (мощность множества/спектра гармоник), M_a — апертура окна аппроксимации ДПГ, K_D — коэффициент редискретизации цифрового сигнала, β — параметр "гладкости" формирующего окна аппроксимации ДПГ.

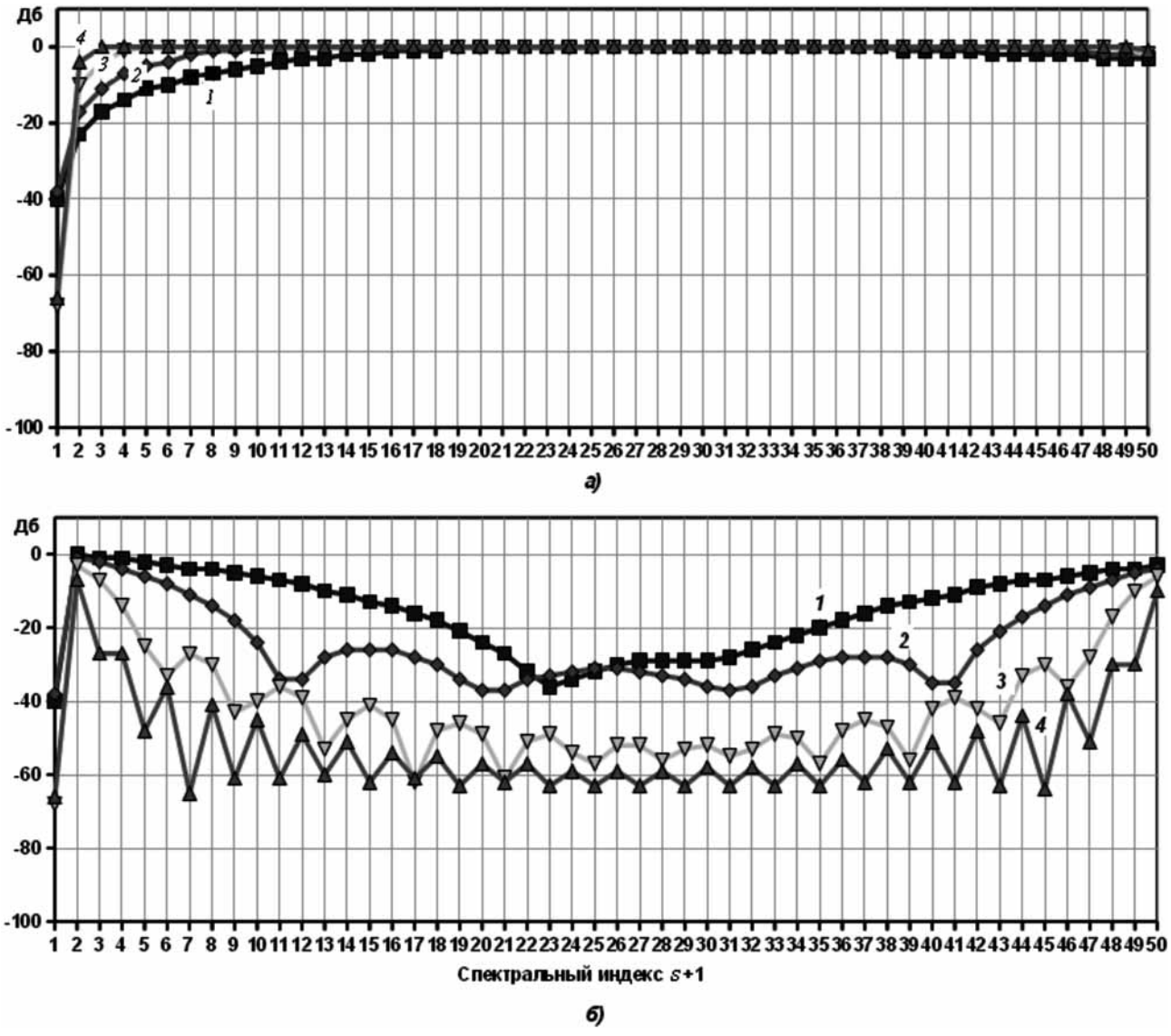


Рис. 2. Графики оценок АЧХ (*а*) и ЧКА (*б*) модели ДПГ (β = 1) при различных значениях апертуры.
Кривые *1—4* соответствуют апертуре M_a = 5, 10, 25, 50 при K_D = 33,3, L_0 = 50, A_0 = 10 000

Результаты отдельных вычислительных экспериментов обычно группируются в серию и в дальнейшем анализируются совместно. Результатом анализа такой серии вычислительных экспериментов может быть новая, часто "формально математически" не вычислимая или "достаточно сложно математически вычислимая" зависимость, в частности, таблица или диаграмма зависимостей оценок ЧКА от значений параметров процедуры ДПГ и/или процедуры синтеза тестовых цифровых сигналов.

На рис. 1—4 показаны графики оценки амплитудно-частотной характеристики (АЧХ) в логарифмическом масштабе и оценки частотной кривой адекватности (ЧКА) для рассмотренных выше моделей ДПГ при различных значениях апертуры окна аппроксимации M_a = 5, 10, 25, 50 и параметра гладкости β = 0, 1, 2. Графики на рис. 1—3 соответствуют коэффициенту редискретизации цифрового сигнала K_D = 33,3, а график на рис. 4 — коэффициенту редискретизации K_D = 1.

Можно отметить, что приведенные кривые нормализованных спектральных погрешностей вычислительной модели ДПГ являются квазипериодическими функциями нормализованной частоты, причем период этих функций зависит от значения апертуры окна аппроксимации. Причина такого характера поведе-

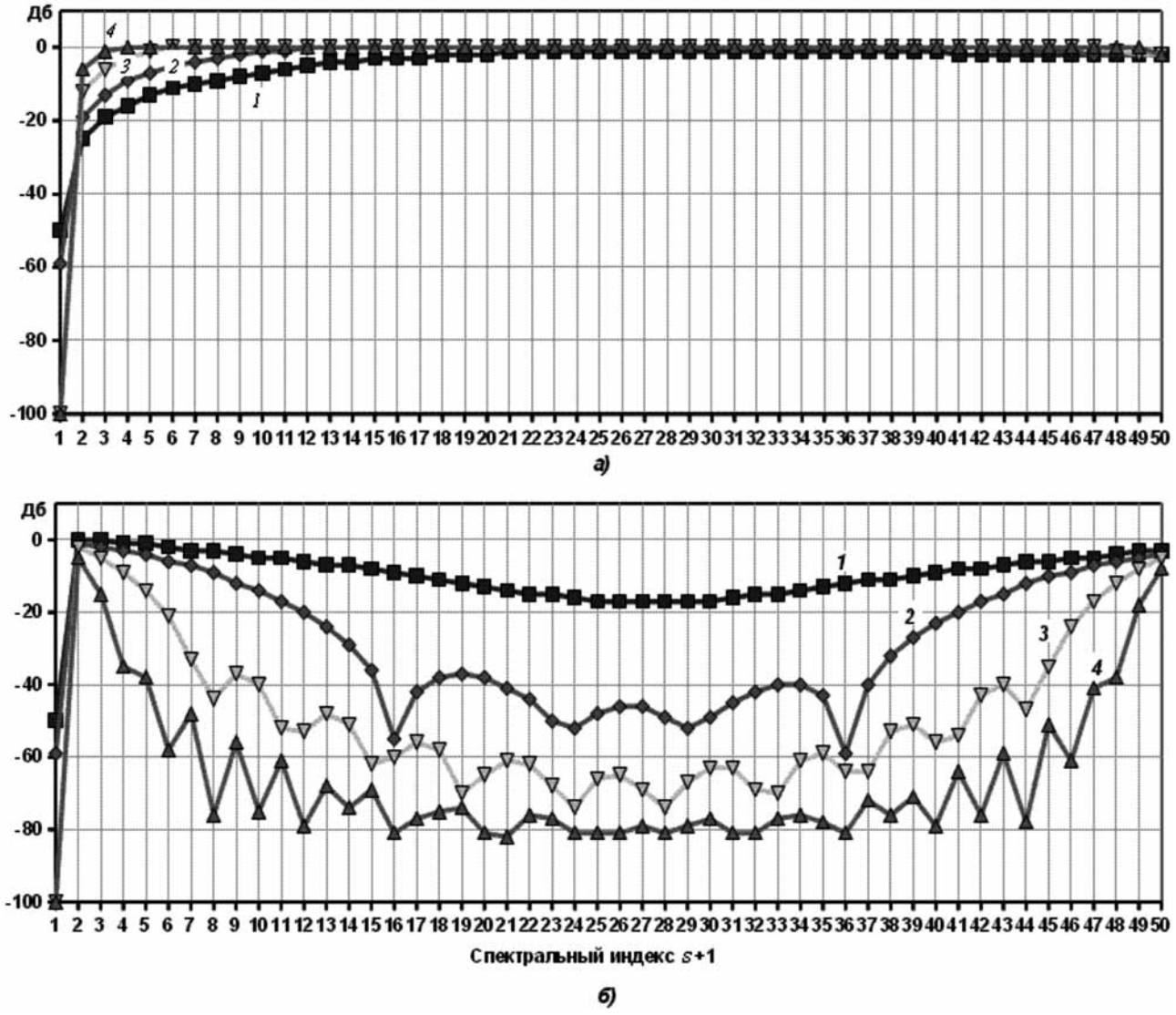


Рис. 3. Графики оценок АЧХ (*а*) и ЧКА (*б*) модели ДПГ (β = 2) при различных значениях апертуры.
Кривые *1—4* соответствуют апертуре M_a = 5, 10, 25, 50 при K_D = 33,3, L_0 = 50, A_0 = 10 000

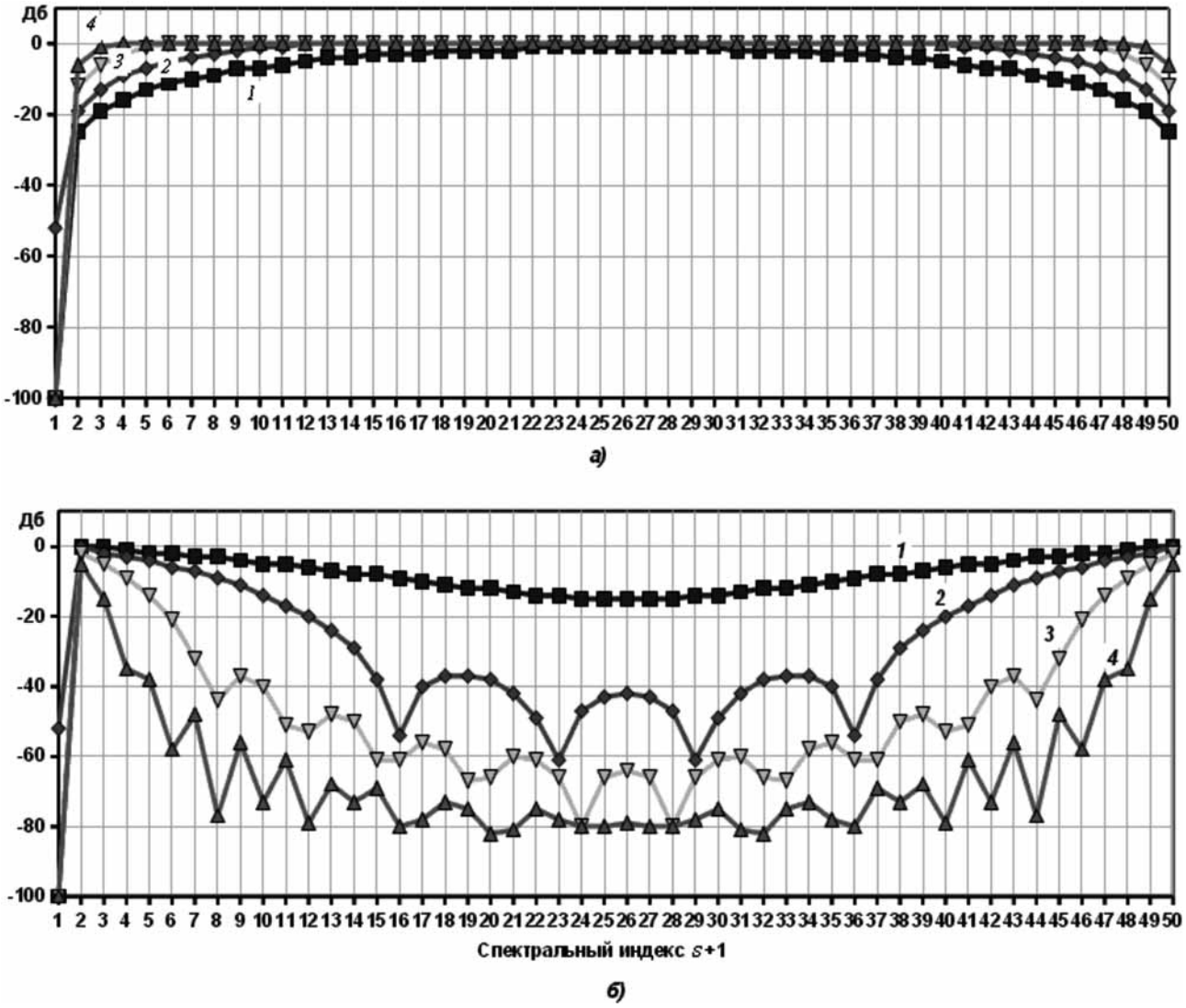


Рис. 4. Графики оценок АЧХ (*а*) и ЧКА (*б*) модели ДПГ (β = 2) при различных значениях апертуры.
Кривые *1—4* соответствуют апертуре M_a = 5, 10, 25, 50 при K_D = 1, L_0 = 50, A_0 = 10 000

ния оценки ЧКА — явление Гиббса, которое возникает при "обрезании" медленно затухающей импульсной характеристики линейного фильтра, в данном случае, ядра преобразования Гильберта.

Использование формирующих окон, повышающих скорость затухания исходной импульсной характеристики линейного фильтра, позволяет существенно уменьшить этот эффект и повысить точность цифрового преобразования в средней полосе спектра частот. Заметим, что обнаружить подобный характер поведения вычислительных моделей ДПГ по графикам оценки АЧХ не так просто.

Сравнивая графики на рис. 3 и 4, нетрудно заметить, что погрешность, обусловленная преобразованием передискретизации цифровых сигналов, не является значительной в полосе низких и средних частот по сравнению с погрешностью, обусловленной ДПГ. Подробное исследование оценки спектральной погрешности редискретизации цифровых сигналов можно найти в работе [4].

Выводы

В последнее десятилетие в области программной инженерии существенно усилилось внимание к средствам повышения качества программного продукта, в частности, к тестированию программного обеспечения (ПО). Если раньше тесты обычно создавали после того, как был написан программный код, то теперь доминирующей тенденцией является движение "*test-*

driven development", основной лозунг которого — сначала тесты, затем код. Наиболее ярким представителем этого движения является Кент Бек — автор методологии экстремального программирования XP [6] и, совместно с Эриком Гамма, стандарта де-факто модульного тестирования (фреймворка JUnit [www.junit.org]), а также известной интегрированной инструментальной среды Eclipse [www.eclipse.org].

Однако при решении задач области знаний "цифровая обработка сигналов (ЦОС)", в отличие от области знаний "коммуникационно-информационные технологии (КИТ)", использование указанных средств тестирования ПО оказывается, вообще говоря, недостаточно. Появляется потребность в дополнительных информационных средствах оценки адекватности используемых вычислительных моделей ЦОС.

В данной работе предложен критерий оценки адекватности класса спецпроцессоров ЦОС, рассмотрен пример задачи ЦОС (вычислительная модель ДПГ) и приведены результаты вычислительных экспериментов по оценке адекватности рассмотренной вычислительной модели. Разработан прототип контрольно-измерительного приложения (КИП) ЦОС.

В отличие от набора модульных тестов, КИП предназначено в большей степени пользователям, а не разработчикам компонентов ЦОС и позволяет заменить довольно значительный объем документации с описанием конкретных метрологических характеристик компонента ЦОС. Можно предположить, что в будущем, с развитием программной инженерии, оценка качества компонентов ЦОС будет включать оценку адекватности реализуемых алгоритмов и соответствующие КИП станут такой же неотъемлемой частью ПО ЦОС как наборы модульных тестов ПО КИТ.

Список литературы

1. **Математический** энциклопедический словарь / Гл. ред. Ю. В. Прохоров. М.: Сов. Энциклопедия, 1988. 847 с.

2. **Самарский А. А.** Компьютеры, модели, вычислительный эксперимент. Введение в информатику с позиций математического моделирования. М.: Наука, 1988. 176 с.

3. **Рабинер Л., Гоулд Б.** Теория и применение цифровой обработки сигналов. М.: Мир, 1978. 848 с.

4. **Чичагов А. В.** Метод оценки качества редискретизации цифровых сигналов // Речевые технологии/Speech Technology. 2009. № 4. С. 26—39.

5. **Чичагов А. В.** Программа оценки точности дискретного преобразования Гильберта (версия 1.0) // Свидетельство РФ о государственной регистрации программы для ЭВМ № 2010614943. Зарегистрировано 29 июля 2010 г.

6. **Бек К.** Экстремальное программирование: разработка через тестирование. СПб.: Питер, 2003. 224 с.

ИНФОРМАЦИЯ

23—24 марта 2012 г. в Москве пройдет очередная конференция

Application Developer Days

Конференция Application Developer Days — это уникальное событие, реализованное экспертами в области программной инженерии, призванное объединить на одной площадке отдельных профессионалов и ИТ сообщества.

Конференция включает в себя обсуждение целого спектра вопросов, связанных с созданием ПО, выбором языков программирования, рассмотрением успешных архитектурных решений и рекомендаций по их созданию, рассмотрением наиболее востребованных технологий, продуктов известных вендоров и Open Source решений.

На конференции планируется обсудить следующие ключевые вопросы:

- Архитектура ПО
- Техники написания кода
- Среды разработки
- Технологии программирования
- Тренды

Контакты:

Республика Беларусь, г. Минск
Владислав Орликов
Телефон: + 375 (29) 770-48-58 (МТС)
Тел./факс: + 375 (17) 504-73-69
ICQ: 71226628
Skype: vldcorp

Россия, г. Ярославль
Андрей Майоров Телефон: +7 903 638 80 68

E-mail: org@it-conf.ru
http://www.addconf.ru/article.sdf/ru/add_3/about

Указатель статей, опубликованных в журнале "Программная инженерия" в 2010–2011 гг.

CONTENTS

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4

Дизайнер *Т.Н. Погорелова.* Технический редактор *Е.М. Патрушева.* Корректор *Е.В. Комиссарова*

Сдано в набор 11.10.2011 г. Подписано в печать 25.11.2011 г. Формат 60×88 1/8.
Усл. печ. л. 5,88. Уч.-изд. л. 7,06. Цена свободная.

Отпечатано в ООО "Белый ветер", 115407, г. Москва, Нагатинская наб., д. 54, пом. 4