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 004.41

В. В. Липаев, д-р техн. наук, проф., гл. науч. сотр.,
Институт системного программирования РАН
e-mail: alexlip@mail.ru

Оценивание количества информации в сложных заказных программных продуктах

Рассматривается взаимосвязь методов и средств, используемых в информатике и экономике, при создании заказных программных продуктов для сложных систем. Предложены методы оценивания количества информации и информационной сложности программных продуктов в системах. Представлены способы оценки и сравнения экономических характеристик интеллектуального труда и сложности проектирования и производства программных продуктов.

Ключевые слова: *заказные программные продукты, количество информации, информационная сложность, интеллектуальный труд, оценка экономических характеристик*

Информатика и экономика программных продуктов заказных систем

Значительная часть проблем в современной ***информатике*** возникает в связи с построением сложных ***программных продуктов для заказных информационных систем***, применяемых в различных областях индустрии и машиностроения [1]. В статье рассматриваются современные программные продукты для систем управления и обработки информации ***в реальном времени, которые*** заказываются и активно применяются в сложных системах динамического управления объектами и процессами. Далее для их обозначения используются термины целевые или ***"заказные"*** системы и объекты управления. К их числу относятся системы в высокоточном технологическом производстве, в авиации, в управлении космическими аппаратами, атомными электростанциями и оборонной техникой. Они являются одними из наиболее сложных ***интеллектуальных систем контролируемого высокого качества***, создаваемых человеком. Их основную, интеллектуальную часть составляют сложные комплексы программ для обработки оперативной информации и управления системами в реальном времени. Эти программы разрабатываются и производятся как промышленные изделия по правилам и стандартам, характерным для производства высококачественной технической продукции. Они требуют большого творческого труда коллективов специалистов, которые применяют ***методы и средства материализации информации о физических объектах и системах.***

Для решения задач создания сложных заказных программных продуктов формируется новая область исследований и проектирования, сочетающая методы и средства ***программной инженерии, включающей экономику*** жизненного цикла программных продуктов. ***Методология программной инженерии*** и международные стандарты регламентируют современные подходы (методы и средства) к производству, внедрению и применению программных комплексов в составе сложно организованных систем. Процессы программирования, верификации, тестирования и сопровождения программных комплексов и их компонентов позволяют получать стабильные, предсказуемые результаты — программные продукты с требуемым качеством обработки технической информации.

Выбор и применение единиц измерения сложности и количества информации в компонентах и комплексах программ — одна из основных задач при анализе и прогнозировании ***экономических характеристик производства*** программных продуктов. Методики определения количественных значений основных информационных и экономических характеристик в проектах различаются и пока не стандартизированы, что вносит дополнительную неопределенность. Для уменьшения неопределенности и сокращения возможных

методических ошибок в информатике и экономике производства комплексов программ представлены ***основные понятия и единицы измерения сложности и количества характеризующей их информации.***

В технических системах философские понятия ***материи, энергии*** и ***информации*** образуют совокупность свойств в создаваемых и применяемых взаимодействующих объектах, процессах или явлениях объективной физической реальности. Информация является одним из основных ***понятий информатики*** — науки о принципах и закономерностях реализации информационных процессов в системах различной природы, отражающих состояние и движение материи или энергии, результатов деятельности человеческого сознания. Объектами информатики являются основные свойства, закономерности преобразования, взаимодействия, хранения, обработки и передачи информации в различных технических системах. ***Количество информации*** является мерой сложности организованных по определенным правилам компонентов систем на базе материи или энергии любой природы. При этом носителем информации могут выступать ***знания о материальном объекте или процессе, а также энергия***, отражающая количество или характеристики определенной информации в системе. Информация является атрибутом функционирования, развития и взаимодействия всех технических систем, объектов и процессов, а также средством сохранения результатов их изменений.

Выбор и применение единиц измерения сложности и количества информации в компонентах и комплексах программ — одна из основных задач при анализе и прогнозировании ***экономических характеристик производства*** программных продуктов. Для уменьшения неопределенности и сокращения возможных методических ошибок в информатике и экономике производства комплексов программ в составе заказных систем должны быть представлены ***основные понятия и единицы измерения сложности и количества характеризующей их информации***, позволяющие провести оценивание и прогнозирование ***сложности и количества соответствующей им информации***. Основной интеллектуальный труд специалистов вкладывается в разработку функциональных алгоритмов и текстов программ. По этой причине необходимо, чтобы выбранные единицы измерения ***количества информации*** в программном продукте были бы в наибольшей степени ***адекватны экономическим показателям их разработки***. Кроме того, должна учитываться не только трудоемкость подготовки алгоритмов и реализующих их отдельных компонентов программ, должна также отражаться трудоемкость их интеграции, создание и реализация комплексной программы испытаний на всех этапах проектирования и производства сложного программного продукта.

Программные продукты в основном определяются ***сложностью их проектирования и производства, а также количеством используемой информации, сопровождающем эти процессы.*** При этом относительно малое значение имеет стоимость средств материализации информации программ в памяти компьютеров. Это позволяет связать измерение количества и сложности информации в таких системах с ***экономическими затратами и количеством квалифицированного труда***, вложенного специалистами в комплексы программ, соответствующие требованиям заказчика. Определяющую роль при этом играет количество интеллектуального труда специалистов, необходимого для ***создания и материализации информации в программном продукте***. Специалисты в области информатики должны уметь применять количественные методы оценки информационных процессов, гарантирующие высокое качество программных продуктов и систем.

Изложенная в требованиях ***информационная сложность комплекса программ должна обеспечивать*** выполнение удовлетворяющих заказчика ***функций, требования*** к программному продукту и разумную вероятность успеха проекта с точки зрения производственных возможностей и квалификации коллектива разработчиков. При оценке трудоемкости разработки комплексов программ, кроме основных интеллектуальных затрат на разработку их функциональных алгоритмов и создание программных компонентов на языке программирования, необходимо учитывать ***затраты труда и количество информации*** на разработку программ их тестирования, на комментарии к программам, на интерфейсы с внешней средой, на сопроводительную документацию. В результате информационная сложность создания программ повышает затраты интеллектуального труда на величину, которая отражает количество информации в функциональных алгоритмах и в текстах программ.

Основной интеллектуальный труд специалистов вкладывается в разработку функциональных алгоритмов и текстов программ. По этой причине необходимо, чтобы выбранные единицы измерения количества информации в программном продукте были бы в наибольшей степени ***адекватны экономическим показателям их разработки***. Кроме того, должна учитываться не только трудоемкость подготовки алгоритмов и реализующих их отдельных компонентов программ, должна отражаться трудоемкость их интеграции, создание и реализация комплексной программы испытаний на всех этапах проектирования и производства сложного программного продукта.

Основные затраты при проектировании и производстве программных продуктов приходятся на ***овеществленный***, преимущественно ***интеллектуальный труд*** специалистов различных категорий. ***Оценивание сложности программного продукта и количества связанной с ним информации,*** используемые для этого понятия и факторы, характерные для рассматриваемого класса заказных систем, ***предлагается*** рассматривать ***в статике***, как де-факто полученные результаты проекти-

рования и производства таких систем, а также ***в динамике*** — в процессе функционирования продукта по назначению.

Информационную сложность процесса проектирования и производства программного продукта для сложно организованной системы можно оценивать:

— по числу строк текстов программного кода — по количеству информации в ***тексте комплекса программ на языке программирования***, которое в значительной степени зависит от конкретных особенностей используемого языка, ориентированного на определенные классы задач, характеризующие ***функциональную, алгоритмическую сложность*** информации для их разработчиков и пользователей;

— по количеству функциональных точек для измерения информационного размера и сложности программных комплексов в группе международных стандартов, создаваемых специалистами функциональных точек, при проектировании и производстве компонентов и программного продукта в целом;

— по экономическим характеристикам — затратам труда коллективов специалистов при полном проектировании и производстве программных продуктов, по оценкам и обобщениям интеллектуального труда, по трудоемкости и длительности создания требуемого количества информации в программном продукте;

— по числу и квалификации специалистов, которые необходимы для проектирования и производства конкретного программного продукта.

Измеряемую трудоемкость организованного проектирования и производства программных продуктов предлагается использовать как ***ориентиры*** при оценивании, сравнении и прогнозировании сложности и количества информации в заказных системах. Это может быть полезно при системном проектировании и сопоставлении эффективности вариантов предлагаемых к реализации программных продуктов. Сравнение на основе различных методов оценивания значений количества информации в комплексах программ позволяет, во-первых, их уточнять и, во-вторых, исключать грубые ошибки в процессе прогнозирования ***экономических характеристик производства при создании*** программных продуктов.

Статическая информационная сложность производства программного продукта отражает количество информации в исходных текстах программ, которые ***анализируются и разрабатываются коллективом специалистов — разработчиков***. Она характеризует сложность, экономические характеристики и трудоемкость создания комплекса программ и его компонентов. Информационная сложность проектирования и производства, в свою очередь, ***определяет динамическую информационную сложность потенциально*** возможных процессов (режимов) функционирования и применения программного продукта в составе прикладных систем. Однако при применении программных продуктов их функциональные возможности, как правило, используются не полностью, что зависит от внешней среды и источников входной информации.

Для каждого проекта программного продукта, выполняющего ответственные функции, при определении и реализации требований контрактов и технических заданий должна разрабатываться и применяться система оценки качества. Необходимо создание и реализация программы, включающей методы и инструментальные средства разработки и испытаний, обеспечивающие ***требуемые качество обработки информации, надежность, безопасность функционирования и применения программных продуктов***. Для этого необходимы трудозатраты ***и другие ресурсы для материализации информации,*** ее практического использования. Методы программной инженерии регламентируют и конкретизируют ***технологические процессы и экономические затраты на обработку информации***, а также на контроль значений качества информации отдельных компонентов на этапах жизненного цикла комплексов программ.

Обычно специалисты — разработчики заказных систем стремятся, по возможности, уменьшить трудозатраты на их эксплуатацию многочисленными пользователями (в динамике), в том числе за счет однократного увеличения информационной сложности проектирования и производства (в статике) комплексов программ. После завершения их производства и испытаний такие компоненты ***отчуждаются от коллектива их создателей***. Они могут применяться, эксплуатироваться, эпизодически модифицироваться и корректироваться. Это приводит к необходимости оформлять достаточно полную ***технологическую документацию***, позволяющую применять, сопровождать и развивать информацию, характеризующую программные продукты различными квалифицированными специалистами. Понятия, измеряемые экономические характеристики, трудоемкость организованного проектирования и производства программных продуктов предлагаются для использования как ***ориентиры*** в процессе оценивания, сравнении и прогнозирования сложности и количества информации в прикладных (целевых) системах.

Простейшие методы оценивания количества информации и информационной сложности программных продуктов

Приступая к разработке комплекса программ, как и в любой производственной деятельности, прежде всего, необходимо провести соответствующее реалиям оценивание исходных данных, характеризующих возможности выполнения проекта. Такая оценка должна ***отражать информационную сложность*** подлежащих разработке программ, реализуемых с их помощью функций, выделяемых для проекта ресурсов и времени. Отсутствие четко декларированных в документах

понятий и требуемых значений характеристик сложности комплексов программ, количества и качества соответствующей им информации, могут вызывать конфликты между заказчиками-пользователями и разработчиками-поставщиками из-за разной трактовки одних и тех же характеристик. Решение задачи определения сложности проекта сводится к заданию базовых требований, включающих соответствующее отдельным компонентам ограниченное число функций, а также требований, намеченных для реализации в конечном программном продукте.

Весьма расплывчатое понятие ***информационная сложность комплексов программ*** и их компонентов значительно конкретизируется и становится измеряемым, когда устанавливается связь этого понятия ***с конкретными экономическими ресурсами, которые необходимы для решения соответствующей задачи*** [2, 3]. ***Сложность или информационный размер комплекса программ*** в настоящее время представляется в проектах в различных единицах, что может изменять их численные значения для одних и тех же комплексов в несколько раз. В качестве таких единиц чаще всего используются: число строк текста программы на языке программирования; количество символов в тексте программы; число байт или команд в объектном коде компьютера после трансляции. Каждая из этих единиц измерения имеет некоторые преимущества при определенных целях исследования и их применения. Однако при сравнительном экономическом и информационном анализе различных проектов, применение в них различных единиц измерения для характеристики объекта производства приводит ***к несопоставимости полученных при этом оценок экономических затрат на выполнение подобных проектов***. Размеры программ, измеренные различными единицами, могут различаться по информационной сложности и экономическим характеристикам из-за наличия между ними промежуточного звена — преобразователя (транслятора) с различными, не полностью определенными характеристиками оптимизации, которые могут изменять оцениваемое количество информации в программном продукте. Это обстоятельство обусловлено, во-первых, особенностями трансляторов, которые преобразуют разработанные специалистом исходные тексты программ в разделы памяти компьютера, заполненные командами, константами или в разделы, выделяемые под данные, и, во-вторых, особенностями языков программирования, структуры и содержания машинных команд.

Размер (объем) текстов программ, размещаемых в памяти компьютера для исполнения, который будем именовать программной сложностью информации, может служить первичным ориентиром при оценке количества информации в комплексе программ. Она влияет на характеристики и стоимость используемых вычислительных машин, которые зависят от объема памяти и производительности компьютера, что особенно важно учитывать в некоторых системах реального времени (например, бортовых, авиационных). Далее в настоящем разделе информационные размеры комплексов программ предлагается определять в терминах ***строк текста программного кода*** на языке программирования (***Lines of code — LOC***) и количества ***функциональных точек*** [2].

Число строк программы на языках высокого уровня (ЯВУ) в значительной степени зависит от конкретных особенностей используемого языка. Языки программирования высокого уровня, как правило, ориентированы на определенные классы задач. Благодаря этому обстоятельству проблемно-ориентированные языки позволяют получать более компактные тексты комплексов программ соответствующих классов. Число строк текста на ЯВУ характеризует ***информационную сложность программы для ее разработчика*** (статическую), однако оно не полностью отражает функциональную, информационную сложность (динамическую).

Использование LOC в качестве единиц измерения информационной сложности программ [2] позволяет получить следующие преимущества:

- возможность сопоставления количества информации в программах и производительности труда в различных группах разработчиков одного проекта комплекса программ;
- возможность применения единиц LOC на промежуточных этапах производства до завершения проекта;
- возможность оценивания размеров компонентов с точки зрения разработчика (физического размера созданного программного продукта, числа написанных и отлаженных строк текста всех компонентов);
- процесс совершенствования программ и информационный размер их расширения может быть сопоставлен с реальным размером компонентов в ходе доработок.

К недостаткам, связанным с применением LOC при оценивании информационной сложности программ, можно отнести следующие:

- единицы измерения LOC трудно применять при прогнозировании информационного размера комплекса программ на ранних стадиях его жизненного цикла;
- разработка программного продукта может быть связана с большими затратами на спецификацию требований и с разработкой сопроводительных документов, которые прямо не зависят от информационных размеров программного кода;
- генераторы (трансляторы) программного кода, не имеющие эффективную оптимизацию, могут создавать чрезмерный размер программного кода, в результате чего искажаются показатели LOC.

Использование функциональных точек в качестве единиц измерения информационной сложности основывается на том факте, что размер программного ком-

понента можно оценивать непосредственно в терминах ***количества информации и сложности функций***, реализованных в данном программном коде, а не только посредством подсчета количества его строк [2]. Целью такого подхода является получение (оценка) единого числа, которое бы полностью характеризовало информационную сложность программного компонента и адекватно соотносилось ***с показателями затрат труда специалистов*** на его создание. Способ определения этой оценки является более эффективным и адекватным трудозатратам на производстве, чем в случае подсчета строк LOC, и позволяет решать ***следующие задачи***:

- оценивать категории сложности компонентов и функций программ;
- разрешать проблемы, связанные с неадекватностью применения единиц измерения LOC на ранних стадиях жизненного цикла комплекса программ;
- определять и учитывать количество и сложность входных и выходных данных, запросов к базе данных, файлов либо структур данных, а также внешних интерфейсов, связанных с программными компонентами и комплексами.

Метод функциональных точек для измерения информационного размера и сложности программных комплексов формализован в группе международных стандартов **ISO 20926** и **ISO 14143:1-5**. В них представлены концептуальные положения методов измерения функциональных точек программных компонентов и перечень требований к стандартизированному методу. В этом методе рекомендуются ***пять базовых элементов*** для определения сложности каждого программного компонента. ***Величина функциональности*** конкретного компонента при этом является взвешенной суммой этих элементов. Такими элементами являются:

- **внешние входы**, которые по-разному влияют на функцию, выполняемую программой;
- **внешние выходы** считающиеся только для существенно различных алгоритмов и сложных функций, выходные данные которых посылаются на существенно разные устройства или компоненты;
- **внешние запросы** — каждый независимый внешний запрос к компоненту рассматривается как отдельный;
- **внутренние логические файлы** — каждая логическая группа пользовательских (системных) данных, которая создается или поддерживается функцией, считается за одну;
- **внешние логические файлы** — каждая логическая группа пользовательских (системных) данных, размещенная во внешних по отношению к приложению файлах, считается за одну.

Число входов и выходов определяет число элементов, направляемых в систему и возвращаемых системой пользователю (компонентам), соответственно. Число запросов — это число выполненных пользователем интерактивных запросов, требующих определенной реакции со стороны программного компонента. Файлы представляют собой любую группу взаимосвязанных элементов информации, поддерживаемую компонентом от имени пользователя (системы), либо предназначенную для организации компонента. Оценка интерфейсов — это число внешних интерфейсов с другими компонентами. Конечной целью функциональной оценки является измерение сложности программного компонента только на основании конкретизированных функций, которые данный компонент должен предоставлять пользователям (системе).

К преимуществам использования функциональных точек в качестве единиц оценивания информационной сложности текстов комплексов программ можно отнести следующие:

- независимость метода от используемого языка программирования и технологии производства программ;
- для пользователей эти единицы измерения могут быть более привычными, при этом облегчается понимание влияния изменений функциональных требований в процессе проектирования и производства на информационные и экономические характеристики программного комплекса;
- возможность измерения и сравнения уровней производительности специалистов при создании комплексов программ, написанных на различных языках программирования или различными коллективами специалистов;
- возможность учета функциональных точек на ранних этапах производства — результаты подсчета функциональных точек могут сравниваться на заключительном этапе.

Недостатки оценивания информационного размера компонентов и особенно крупных комплексов программ методом функциональных точек:

- использование субъективных оценок размеров и сложности функциональных точек, вследствие чего возможны конфликты между заказчиком и разработчиками при оценке стоимости и производительности отдельных специалистов, участвующих в реализации проекта;
- зависимость получаемых результатов информационных размеров и сложности программных продуктов от технологии, используемой для их оценки;
- необходимость преобразования количества функциональных точек в значения LOC вследствие зависимости некоторых моделей оценки в проектах трудозатрат от показателя LOC.

Анализ и оценка функциональных возможностей разрабатываемых компонентов позволяет рассчитывать и сравнивать месячную ***индивидуальную производительность труда*** определенного специалиста или целой рабочей группы. Следует заметить, что с позиций традиционно используемых подходов ***взаимодействие производителям с заказчиком*** при расчетах информа-

ционной сложности и стоимости компонентов более удобно. Однако при коллективном производстве сложных программных продуктов индивидуальные характеристики специалистов нивелируются, многие из них не участвуют в непосредственном программировании (аналитики-спецификаторы, тестировщики, документаторы). Как следствие, расчет функциональных возможностей и его ***применение в экономике производства сложных программных продуктов не всегда эффективно***.

Оценивание информационной сложности программных продуктов на основе экономических характеристик

Основной недостаток представленных выше методов оценивания информационной сложности и трудоемкости производства комплексов программ состоит в акценте на затраты труда только на создание текстов функциональных программ. Однако при производстве сложных программных продуктов практически столь же большой труд вкладывается специалистами в процессы тестирования, документирования, сертификации и различные вспомогательные работы. Обобщенно все количество информации в целостном программном продукте определяется ***экономическими характеристиками — энергией труда специалистов*** при полном его проектировании и производстве с учетом технического вспомогательного труда. Эти характеристики были получены, обобщены и изучены на основе экспериментальных данных реальных проектов. ***Статистические исследования экономических характеристик*** большой совокупности (сотни) завершенных программных продуктов обеспечили возможность учета полных затрат интеллектуального труда, которые адекватны их информационной сложности. На основе этих данных созданы несколько математических моделей для получения оценок трудоемкости, длительности и других экономических характеристик производства программных продуктов. Наиболее известной и широко используемой является модель COCOMO II [2, 3], основные особенности которой изложены ниже.

При увеличении объема кода комплекса программ установлен более быстрый рост информационной сложности их разработки, чем тот, который отражается простой линейной зависимостью от объема кода. Эксперементально подтверждено, что по мере увеличения базового фактора — размера комплекса, возрастает относительная трудоемкость разработки каждой строки в программе. Для учета влияния на трудоемкость основных факторов предложено использовать ***коэффициенты (рейтинги) изменения трудоемкости*** (КИТ) производства, учитывающие влияние каждой составляющей совокупных затрат. В них входят: факторы, характеризующие процесс непосредственной разработки; факторы программной и аппаратурной оснащенности; квалификация участвующих в разработке специалистов. Накопленный опыт и статическое обобщение проведенных исследований уже выполненных проектов позволили детализировать влияние ***четырех основных групп факторов***, определяющих экономические характеристики при проектировании и производстве программных продуктов:

- факторы, отражающие ***требования к характеристикам функций и качества*** создаваемого комплекса программ как объекта проектирования и производства;
- факторы, определяющие ***организацию коллектива*** для производства программного продукта и его обеспечения квалифицированными специалистами;
- факторы, характеризующие ***технологическую среду*** и оснащенность инструментальными средствами автоматизации процесса производства программного продукта;
- факторы, отражающие оснащенность процесса производства программного продукта ***аппаратурными вычислительными средствами,*** на которых реализуется программный продукт и базируются инструментальные системы автоматизации разработки.

Наиболее полные статистические исследования трудоемкости и затраты времени на создание сложных программных продуктов обобщены и отражены в аналитических моделях [2]. При определении коэффициентов в модели COCOMO II за начало разработки комплекса программ принят момент создания технического задания, а за окончание — завершение испытаний программного продукта в целом. Диапазону размеров современных программных продуктов в три-четыре порядка (до 10 млн строк) соответствуют приблизительно такие же диапазоны изменения трудоемкости и стоимости производства программных продуктов. Однако вариации продолжительности времени производства программных продуктов значительно меньше, чем изменения трудоемкости их разработки. Временные вариации при этом не превышают диапазон от нескольких месяцев до 4—5 лет. Интервал времени на разработку ограничен сверху и снизу, и одним из основных факторов, определяющих эти границы, является размер комплекса программ.

Для ***прогнозирования трудоемкости производства программных продуктов*** (человеко-месяцы) ***в модели COCOMO II*** [3] предложены выражения, учитывающие влияние 22 факторов в базовых значениях размера комплекса программ. При создании многих сложных продуктов затраты исчисляются сотнями человеколет, что определяет особую актуальность более точного учета факторов, способствующих снижению прогнозируемых затрат труда специалистов при таких масштабах производства и применения. По этой причине зачастую необходим предварительный ***системный анализ*** и более точное распределение ресурсов на этапы производства конкретных комплексов программ путем выбора КИТ с учетом всего их жизненного цикла.

При использовании выражений для прогнозирования экономических характеристик проектов следует выбирать набор факторов (***калибровать модель***), имеющих определяющее значения КИТ в соответствии с характеристиками ***конкретного проекта***.

Большую роль в снижении информационной сложности и повышении экономической эффективности производства программных продуктов играет ***повторное использование готовых программных компонентов*** из других проектов. При этом необходимо учитывать дополнительные затраты, которые необходимы для того, чтобы обеспечить возможности повторного использования компонентов (ПИК) вместо разработки таких компонентов и, как следствие, сократить интегральные затраты на производство продукта. Для оценки и учета сокращения трудоемкости и длительности производства за счет повторного использования готовых компонентов в модели COCOMO II рекомендуются соответствующие дополнительные выражения и процедуры. При прогнозировании экономических характеристик предлагается в качестве дополнительных данных учитывать долю затрат на программный продукт, которая может быть сокращена за счет применения модифицируемых и/или полностью готовых ПИК.

Длительность разработки программных продуктов является важнейшей экономической характеристикой, поскольку зачастую именно она определяет общие сроки разработки сложноорганизованных заказных систем, для которых эти продукты разрабатываются. На основе оценки значений трудоемкости, размера комплекса программ и выбранных значений факторов в модели COCOMO II могут быть рассчитаны ***длительность*** производства и необходимое для его реализации число ***специалистов.*** При этом рекомендуется учитывать те же наборы факторов, которые применяются при прогнозировании трудоемкости производства программного продукта.

Установлено, что длительность производства комплексов программ меньше подвергается изменениям при автоматизации разработки или применении других методов ее улучшения, чем трудоемкость или производительность труда специалистов. Необходимость выполнения определенной совокупности этапов и операций в заданной технологической последовательности в процессе производства, как правило, изменяют незначительно при различных воздействиях на процессы разработки. Исключением является применение ПИК, при котором значительно сокращаются этапы программирования и автономного тестирования модулей и компонентов программ, а также в той или иной степени длительность других этапов. По этой причине зависимость длительности производства от доли ПИК оказывается нелинейной. Заметное сокращение времени разработки проявляется только при создании версии программного продукта из практически полностью готовых компонентов.

Преимущества модели COCOMO II состоят в следующем:

- возможен учет достаточно полной номенклатуры факторов, влияющих на оценивание экономических характеристик и информационной сложности производства сложных программных продуктов;
- фактические данные из таблиц подбираются в соответствии с реальными комплексами программ, которые могут соответствовать конкретному проекту и предприятию;
- метод является достаточно универсальным и может поддерживать различные размеры, режимы и уровни качества продуктов;
- возможна высокая степень достоверности результатов калибровки модели с опорой на предыдущий опыт коллектива специалистов;
- результаты прогнозирования сопровождаются обязательной документацией;
- модель относительно проста в освоении и применении.

К недостаткам модели COCOMO II можно отнести следующие:

- все результаты использования модели зависят от размера программного текста продукта, как следствие, точность такой оценки оказывает ***определяющее влияние*** на результаты прогноза информационной сложности, трудозатрат, длительности разработки продукта, численности участвующих в разработке специалистов и производительности их труда;
- недостаточно полно учтено влияние требований функциональной безопасности программного продукта;
- не учитывается зависимость между интегральными затратами труда и количеством времени, затрачиваемым на каждом этапе проекта;
- недостаточно учитывается внешняя среда в процессе производства и применения программного продукта.

Кроме того, модель COCOMO II представляет ***трудозатраты по сумме всех этапов производства*** (от этапа разработки концепции до этапа поставки программного продукта). При прогнозировании экономических характеристик производства программных продуктов ***для систем реального времени*** необходимо учитывать затраты на создание моделей внешней среды. Такие затраты по величине могут быть сопоставимыми с затратами на основной программный продукт. Приведенные недостатки и трудности применения модели COCOMO II для прогнозирования информационной сложности и экономических характеристик производства сложных программных продуктов в значительной степени могут быть скомпенсированы в ходе приобретения квалификации и ***накоплении опыта*** специалистами, которые активно занимаются задачами оценивания в конкретной прикладной области.

Динамическая информационная сложность функционирования программного продукта в составе систем

Информационную сложность процессов функционирования программного продукта в составе заказных систем в реальном времени определяют:

- количество информации в программном продукте, необходимое для его ***функционирования*** по назначению в единицу времени;
- количество информации, ***поступающее от источников*** на вход программного продукта заказной системы в единицу времени;
- количество достоверной информации ***для потребителей на выходе*** программного продукта заказной системы в единицу времени;
- требуемая функциональная надежность и безопасность применения информации, которая соответствует программному продукту в составе прикладной системы.

Динамическая информационная сложность отражается на длительности исполнения комплекса программ или на времени обработки на компьютере различных видов информации, включая исходные данные, которые необходимы для получения ***требуемых информационных результатов***. На них может влиять объем информации характеризующей программы, ***данные, которые, размещаются в реализующем (объектном) компьютере, а также*** объем памяти и производительность компьютера, необходимые для корректного функционирования и исполнения программного продукта при его применении пользователем или некоторой внешней системой. Для применения и эксплуатации программного продукта основными определяющими ресурсами могут быть память и производительность реализующего компьютера. В качестве ограничений при этом могут выступать допустимое (приемлемое) время счета задач, а также доступный для базы данных объем памяти.

В сложных ***системах реального времени*** их информационная, функциональная сложность непосредственно отражается на требуемом заказчиком ***времени отклика*** (ожидания выходных результатов) при определенных исходных данных. Пропускная способность комплекса программ при его исполнении на конкретном компьютере характеризует ***динамику обработки информации*** и ***информационную сложность исполнения программ*** в реальном масштабе времени. Информационная сложность решения задачи может изменяться в зависимости от имеющихся ресурсов компьютера, структуры и типов исходных данных. Требования минимизации допустимого времени отклика (времени получения результатов) и ресурсов производительности компьютера в реальном времени могут приводить к значительному повышению информационной сложности решения задач. В этом случае задача тем сложнее, чем ***быстрее необходимо получать результаты*** при реальной ограниченной производительности вычислительных средств. В процессе функционирования и эксплуатации комплекса программ основные сложности могут быть сосредоточены в количестве информации, включая исходные данные, а также при анализе, обработке и применении результатов.

Сложность подготовки, корректировки и контроля информации больших массивов данных может значительно превышать программную или временную информационную сложность функционирования программного продукта и технической системы.

В развитии ***динамической информационной сложности современных программных продуктов*** существуют следующие тенденции:

- непрерывно увеличивается средняя сложность производства и количество информации в программных продуктах систем управления и обработки информации;
- обычно в таких системах и программных продуктах для выполнения требуемых заказчиком функций сокращаются сложность и количество информации, поступающей от пользователей и из внешней среды;
- уменьшаются сложность и количество обработанной информации, выдаваемой внешним потребителям и системам — упрощается управление заданными функциями программных продуктов.

* * *

Оценивание и прогнозирование сложности и количества информации в программных продуктах технических систем — это универсальный метод обобщения их экономических характеристик. Это позволяет связывать количество и сложность информации в системах с объемом квалифицированного интеллектуального труда, вложенного специалистами в комплексы программ, соответствующие требованиям к ним заказчика. Сравнение значений количества информации, определенных различными методами оценивания, способствует сокращению ошибок в процессе прогнозирования экономических характеристик производства при создании программных продуктов.

Список литературы

1. **Колин К. К.** Философские проблемы информатики. М.: БИНОМ. Лаборатория знаний, 2010. 264 с.

2. **Липаев В. В.** Экономика производства программных продуктов. Издание второе. — М.: СИНТЕГ. 2011. 352 с.

3. **Boehm B. W.** et al. Software cost estimation with COCOMO II. New Jersey: Prentice Hall PTR, 2000.

УДК 004.89; 004.416; 004.412

Е. А. Шалфеева, доц., стар. науч. сотр.,
ИАПУ ДВО РАН,
e-mail: shalf@iacp.dvo.ru

Мониторинг информационных ресурсов жизнеспособной интеллектуальной программной системы

Представлен спектр подзадач мониторинга ресурсов, необходимых для управления интеллектуальными программными системами и предложен метод такого мониторинга.

Ключевые слова: *интеллектуальная программная система, решатель задач, информационный ресурс, онтология, модели программного обеспечения, проектная модель программы, графовая модель, структурное свойство, метрика проектной модели, инженерия интеллектуальных программных систем*

Введение

Проблема разработки практически полезных, реально и широко используемых интеллектуальных программных систем (ИПС) является одной из центральных проблем в области искусственного интеллекта. Согласно работам [1, 2] новые идеи в сфере создания жизнеспособных ИПС связаны с *управлением* ими. Термин "сопровождение программного средства", под которым понимается изменение кода программы, заменяется термином "управление программным средством", под которым понимается решение задач сопровождения программного средства с помощью специальных высокоуровневых механизмов управления, сводящих к минимуму изменение его кода [1, 3, 4].

Управляемость программы в процессе ее жизненного цикла может быть достигнута сведением к некоему разумному минимуму ее процедурного компонента за счет представления как можно большей информации в ее декларативном компоненте [1, 2]. В данной работе программная часть ИПС представляется как семантическая сеть, с терминальными вершинами которой связаны программные единицы (ПЕ), содержащие программный код. В последнее десятилетие близкие идеи используются в области интеграции отдельных программ в программные системы ("оркестровка" сервисов) с использованием декларативного представления схемы интеграции [5, 6] и в области генерации кодов программ по UML-моделям с помощью инструментария для разработки под управлением моделей (MDD) [3, 4]. Декларативные компоненты рассматриваются как "рычаги" для получения управляющих воздействий, обращенных к ИПС как объекту управления.

При разработке и управлении ИПС используются также модели предметной области и решаемых задач, их онтологии [1, 7], а также документация ИПС, компоненты которой структурированы и доступны для редактирования. Важнейшими представителями ИПС являются системы, в архитектуре которых явно выделена отдельно хранимая, пополняемая и редактируемая база знаний предметной области. Базы знаний, базы данных, а также онтологии являются информационными ресурсами (ИР), которые могут обрабатываться интеллектуальными системами. Некоторые из онтологий не используются непосредственно, но являются метаинформацией для знаний либо данных. Качество всех этих ИР влияет на качество ИПС и требует соответствующего управления и мониторинга, понимаемого в широком смысле как "специально организованное, систематическое наблюдение за состоянием объектов ... с целью их оценки, контроля или прогноза" (согласно www.tracker.co.ua/gps_glossary.html).

Цели настоящей работы — описать спектр подзадач мониторинга ресурсов, необходимых для управления ИПС, и предложить метод такого мониторинга.

Задачи управления ИПС, требующие мониторинга ИР

Использование при реализации интеллектуальной системы с концептуальной базой знаний какой-либо универсальной машины вывода оказывается невозможным. В этом случае самостоятельным компонентом интеллектуальной системы становится *решатель задач* (рис. 1), т.е. программа, которая, используя базу знаний, решает задачи интеллектуальной системы [1].

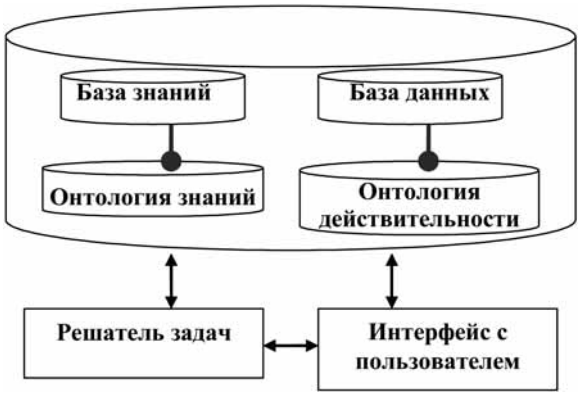

Рис. 1. Основные компоненты ИПС

Основные управляющие воздействия состоят в изменении базы знаний и изменении моделей решателя (рис. 2), приводящих к изменению функциональности и других характеристик ИПС. Целью управления решателями задач является перманентное приведение их свойств в соответствие с изменяющимися текущими требованиями, условиями эксплуатации, знаниями предметной области, а также возможными изменениями онтологии предметной области [1].

На сегодняшний день очевидна необходимость и возможность управления ИПС на уровне проектных моделей приложения (решателя ИПС), на уровне проектных моделей отдельной ПЕ (создаваемой в рамках приложения, но претендующей на последующее использование другими приложениями) и на уровне представления функциональности приложения (модель обрабатываемых данных, модель подзадач и их связи с данными).

Поскольку модели каждого из этих уровней представимы семантическими сетями, то основной набор операций (управляющих воздействий) похож на операции работы с сетевыми моделями данных.

Операции управления на уровне представления функциональности таковы: *добавить вершину-подфункцию* в модель подзадач приложения, *удалить вершину-подфункцию* из модели подзадач приложения, *изменить название функции*. В модели обрабатываемых данных и их связей с подзадачами в качестве *входных или выходных ресурсов приложения* рассматривается их *метаинформация (онтология),* поэтому требуются операции над онтологией: *удалить, добавить, изменить элемент (вершину) онтологии ресурса, добавить или удалить связь (дугу) подфункции с онтологией ресурса.*

Операции управления на уровне проектных моделей приложения таковы: *добавить ПЕ в архитектурное представление* (сеть), *удалить ПЕ* из сети, *изменить имя* (ссылку на) ПЕ, *добавить связь* между существующими ПЕ, *удалить связь* между ПЕ.

Для ИПС, решатели задач которых проектируются как многоагентные приложения (в рамках экспериментальной технологии разработки ИПС, исследуемой в лаборатории Интеллектуальных Систем ИАПУ ДВО РАН), требуются дополнительно операции для "более тонкого" управляющего воздействия. То же относится и к управлению на уровне проектных моделей такой ПЕ (агента).

Более подробное рассмотрение этих подзадач управления структурировано следующим образом: суть управления, требующийся мониторинг ИР, требуемое управление документацией ИПС.

Подзадачи управления на уровне представления функциональности приложения

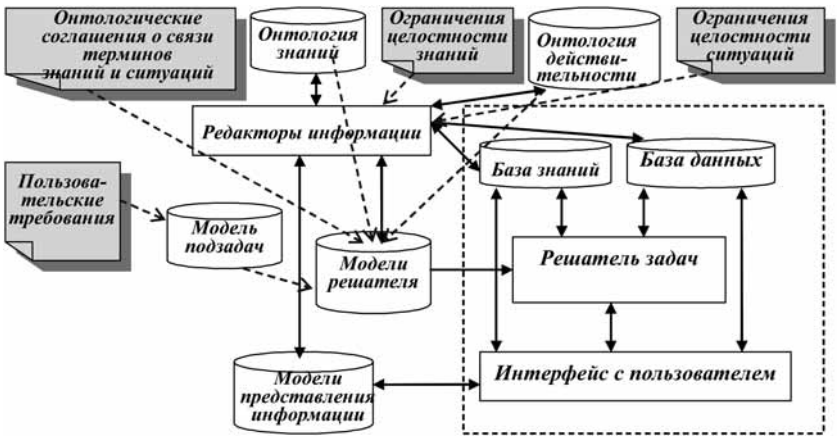

Рис. 2. Информационные ресурсы, используемые при разработке ИПС

В рамках экспериментальной технологии разработки ИПС проектируются ПЕ, которые абстрагированы от знания того, кому они передают управление, они лишь имеют знание о том, что они должны сообщить, т.е. знают формат для связи с неизвестным получателем управления (сообщения).

Связывание этих ПЕ (агентов, отдельные блоки которых настроены на обработку разных типов сообщений), т.е. их "оркестровка" осуществляется в архитектурной модели — сети, в которой узлы соответствуют блокам (продукций агентов), а дуги помечены шаблонами передаваемых сообщений и уточняющими метками.

Добавление подзадачи в приложение (без изменений входных или выходных данных)

Этот случай означает изменение в *модели подзадач приложения (добавление одной вершины-подзадачи* и требуемых связей с метаинформацией обрабатываемого данного), обусловленное либо более детальным анализом функциональности, либо расширением требований к приложению.

Управление состоит, например, в подборе готовой ПЕ (агента), реализующей эту подфункцию, и (при ее наличии) добавления ее в архитектурный проект для последующей отладки.

Мониторинг: поиск *обрабатывающих агентов*, предназначенных (согласно их спецификациям) для обработки данных с той же метаинформацией, что и у подзадачи, а при отсутствии обрабатываемых данных — поиск *управляющих агентов*; оценивание морфологических свойств модели подзадач, длины и уникальности названий вершин-подзадач.

Управление документацией состоит в отслеживании изменений в следующих документах: Требования, Архитектурный проект (приложения), Руководство по использованию приложения, Тесты приложения.

Удаление подзадачи из приложения

Управление состоит в удалении лишней подфункции из функциональности приложения.

Мониторинг: *проверка неизбыточности ИР для множества подзадач* (требуется выявить фрагменты онтологий входных и выходных ресурсов, для которых не предусмотрена обработка подзадачами приложения, а именно: требуется выявить вершины в метаинформации ИР, такие, что ни на одну вершину "растущего" из нее поддерева нет ссылки из вершины-подзадачи в *модели подзадач приложения*).

Управление документацией состоит в отслеживании изменений в следующих документах: Требования, Архитектурный проект, Руководство по использованию приложения, Тесты приложения.

Расширение онтологии входного или выходного данного (чаще вместе с добавляемой подзадачей)

В этом случае, возможно, потребуется внести изменения в *модель подзадач приложения*, чтобы для добавляемого элемента в структуру информации была предусмотрена обработка в какой-нибудь подзадаче приложения.

Управление состоит, например, в выборке ПЕ (*агентов)*, включенных в архитектурный проект приложения, и работающих с обновленными ИР, в планировании создания новых или изменении существующих ПЕ (блоков), работающих с обновленными ИР.

Мониторинг: *проверка соответствия* между *онтологией ИР и моделью подзадач приложения* (при добавлении вершины в метаинформацию входного или выходного ресурса требуется проверить, понадобятся ли данные такого типа некоторой подзадаче приложения, иначе говоря, требуется выявить вершины или поддеревья *онтологии*, с которыми не связаны вершины-подзадачи в *модели подзадач приложения*).

Управление документацией состоит в отслеживании изменений в следующих документах: Структура входных/выходных ресурсов, Модель подзадач решателя, Архитектурный проект, Проект агента, Тесты приложения.

Изменение онтологии входных или выходных данных

Этот случай означает изменение сорта или названия некоторой вершины метаинформации входного или выходного ресурса приложения, удаление некоторой вершины метаинформации входного ресурса. В случае входных данных, возможно, потребуется внести изменения в *Проект агента*, так как скорее всего, изменится *операция проверки целостности ИР.*

Управление состоит, например, в выборке ПЕ (*агентов*), включенных в архитектурный проект приложения, работающих с ИР, созданными по данной метаинформации, в определении тех из них, на которые влияет это изменение; в планировании создания новых или изменения существующих блоков агентов.

Мониторинг:

- при изменении сорта или названия некоторой вершины метаинформации входного ресурса приложения может потребоваться *найти все порожденные по этой вершине элементы* входного ресурса (если он уже порожден), чтобы кто-то проверил, не "пострадали" ли они;
- при удалении некоторой вершины метаинформации входного ресурса приложения требуется найти все связанные с удаляемой вершиной подзадачи приложения (в *модели подзадач приложения*), они станут кандидатами на удаление; требуется найти и проверить все связанные с поддеревом, в котором удаляется вершина, подзадачи приложения (*в модели подзадач приложения*).

Управление документацией состоит в отслеживании изменений в следующих документах: Требования, Архитектурный проект, Проект агента, Тесты приложения.

Подзадачи управления на уровне проектных моделей приложения

Изменение в архитектурном проекте приложения

Управление состоит, например, в замене версии агента, в замене блока — получателя некоторого сообщения.

Мониторинг:

- *проверка внутренней согласованности архитектурного проекта приложения* (требуется выявить сообщения, которые некому передать или не от кого получить, а именно: требуется выявить дуги передачи сообщения в сети (*архитектурном проекте*) без вершины-начала или без вершины-конца);
- *проверка согласованности архитектурного проекта приложения* и *проектов агентов* (требуется выявить наличие блоков у каждого агента для обработки сооб-

щений, передаваемых в соответствии с *архитектурным проектом* приложения).

Управление документацией состоит в отслеживании изменений в следующих документах: Модель подзадач решателя, Спецификация агента, Проект агента, Тесты приложения.

Добавление ПЕ в архитектурную модель приложения

Управление состоит, например, в добавлении агента или нового блока некоторого агента, и "настройке" на получение некоторого сообщения добавленному блоку.

Мониторинг:

- *проверка внутренней согласованности архитектурного проекта приложения* (требуется выявить сообщения, которые некому передать или не от кого получить, а именно: требуется выявить дуги передачи сообщения *архитектурного проекта* без вершины-начала или без вершины-конца);
- *проверка согласованности архитектурного проекта приложения* и *проектов агентов* (требуется выявить наличие блоков у каждого агента для обработки сообщений, передаваемых в соответствии с управляющим графом приложения, а именно: требуется выявить пары <название шаблона, имя агента> для вершин-блоков, таких, что в проекте агента, имя которого совпадает с "имя агента", нет вершины "блок для шаблона", имя которой совпадает с "название шаблона").

Управление документацией состоит в отслеживании изменений в следующих документах: Модель подзадач решателя, Спецификация агента, Проект агента, Тесты приложения.

Подзадачи управления на уровне проектных моделей агентов

Добавление или удаление блока

Управление состоит в изменении числа блоков некоторого агента, например, в целях усовершенствования его как повторно используемой ПЕ.

Мониторинг:

- *проверка соответствия* подзадач (в *модели подзадач приложения*) и блоков агентов;
- *проверка соответствия* вершин *архитектурного проекта приложения блокам агентов* приложения.

Управление документацией состоит в отслеживании изменений в следующих документах: Архитектурный проект, Спецификация агента, Руководство по использованию агента, Тесты агента.

Изменение блока

Управление состоит в изменении, например, формата получаемых или передаваемых сообщений или изменении метода решения (алгоритма) назначенной блоку подфункции.

Мониторинг: *проверка неизбыточности и достаточности совокупности ПЕ для обрабатываемых ИР* (требуется выявить онтологии входных и выходных ресурсов, соответствующих этому блоку подзадач в *модели подзадач приложения,* и убедиться в их достаточности и отсутствии лишних).

Управление документацией состоит в отслеживании изменений в следующих документах: Архитектурный проект, Спецификация агента, Руководство по использованию агента, Тесты агента.

Метод мониторинга ИР

Предлагаемый метод мониторинга ИР основан на определении и получении значений структурных свойств таких ресурсов, как хранимые в виде семантических сетей декларативные компоненты, модели, компоненты документации и обрабатываемые знания и данные.

Если под *свойством ресурса* понимать некоторую его характерную особенность или особенность некоторого его осмысленного фрагмента, то *структурным свойством ресурса* является свойство, определяемое только на основе анализа структуры этого ресурса [8, 9].

В рамках единого подхода к оцениванию структурных свойств для произвольного информационного ресурса предложены *графовые модели*, отражающие различные структурные свойства онтологий и других информационных ресурсов [8, 10]. Структурный подход позволяет единообразно отражать и измерять внутренние свойства ресурсов. Для определения структурных свойств используются некоторые из общего множества *графовых моделей* [8]. Все графовые модели являются ориентированными размеченными мультиграфами: и вершины, и дуги в общем случае имеют имя, метку и принадлежат к одному из задаваемых типов. Для каждой из множества *графовых моделей* описан способ ее построения по формализованному представлению информационного ресурса. Примерами графовых моделей одного ресурса являются *граф "теоретико-множественных" связей, граф стандартной партономии, граф предметно-ориентированных связей* [8, 10].

Примеры подзадач мониторинга архитектуры приложения (моделируемой графом предметно-ориентированных связей):

- *найти множество вершин, в которые не входит ни одна дуга* передачи сообщения*;*
- *найти совпадающие метки дуг;*
- *найти множество вершин, у которых нет связи "ссылка на агента".*

Примеры подзадач мониторинга *метаинформации входного ресурса* приложения (моделируемой *графом партономии*):

- *найти вершины-нетерминалы, для которых все вершины-листы необязательны*;
- *найти пути в онтологии входного ресурса с недоопределенными терминами.*

Примерами графовых моделей совокупности ресурсов (или разделенного на модули ресурса) являются *граф использования* и *граф включения* [8].

Граф использования применяется для мониторинга так называемых *"горизонтальных" связей*, т.е. связей между онтологиями (связей использования из данной онтологии к другим онтологиям), между компонентами документации и декларативными моделями.

Примеры подзадач мониторинга согласованности архитектуры приложения и *модели подзадач* ("горизонтальная "связь" декларативного компонента приложения и компонента документации приложения):

- *найти множество вершин-подзадач* (в *модели подзадач приложения*), *метки которых не совпадают с именами вершин-агентов* (архитектурного проекта *приложения);*
- *найти множество вершин-блоков в архитектурном проекте, для которых нет связи с вершиной "ссылка на агента" либо для вершины "ссылка на агента" не определено значение.*

Граф включения применим для исследования *"вертикальных" связей* — связей соответствия хранимых ресурсов своим онтологиям.

Примеры подзадач мониторинга согласованности ИР с их онтологией:

- *найти число дуг в метаинформации входного ресурса, по которым еще не порождены вершины-термины в данной информации;*
- *найти число информационных ресурсов, порожденных по данной метаинформации;*
- *найти дуги в метаинформации, для которых ни в одном информационном ресурсе не порождены элементы информации.*

Управление ИР

В ИР важно содержание информации и ее удобное представление для пользователей, а также соответствие постоянно меняющимся априорным знаниям пользователей. Поскольку концептуальный ИР представляется в терминах соответствующей онтологии, он является "неразрывной парой", состоящей из собственно содержания этого ресурса и его онтологии, которые находятся между собой в определенном соответствии. Онтология ИР формируется инженером знаний, а его содержание — носителем соответствующей информации [1].

Мониторинг информационных ресурсов направлен на получение неявной информации, содержащейся в них. В случае, когда мониторинг осуществляется в соответствии с априорными знаниями специалистов, этими знаниями могут быть, например, явные определения структурных моделей этих ресурсов, их структурных свойств в терминах этих моделей, дефектов, недостатков, неудобств и особенностей этих ресурсов в терминах этих свойств. *Задачами мониторинга* информационных ресурсов в этом случае являются построение на основе априорных знаний структурных моделей этих ресурсов, определение значений их структурных свойств, выявление дефектов, недостатков, неудобств и особенностей в них [1, 9].

При мониторинге ресурсов наиболее важно обнаружить множество *дефектов*. Немалое значение для качества ИПС имеют и *потенциальные неудобства*.

Дефектом информационного ресурса считаем свойство (или состояние) ресурса, делающее его непригодным для использования при разработке ИС. **Потенциальное неудобство** — свойство ИР, затрудняющее возможность человека работать с информацией или снижающее эффективность его автоматической обработки [9].

Структурными дефектами базы знаний являются *наличие цикличных утверждений* (определяется как множество или число циклов в графовой модели), а также *несвязанность элементов знаний* (определяется как отсутствие дуг в графовой модели). На практике это редко встречающиеся дефекты.

Более распространенными являются неструктурные дефекты — показатели неполноты, некорректности. К таковым относятся: *отсутствующие фрагменты (элементы) знаний*, *неверные названия терминов, неверные значения сущностей*. Автоматизация их поиска трудноосуществима. Однако анализ структурных свойств позволяет найти "подозрительные места" в информационном ресурсе, указать на структурные особенности, повышающие риск внесения ошибок и риск их невыявления и обеспечить демонстрацию фрагментов структуры, облегчающую поиск ошибок экспертом [9]. Для выявления "подозрительных мест" эффективно использовать следующие метрики:

- *список всех совпадающих терминов*;
- *список всех похожих терминов* (возможные критерии: более половины букв от начала названия, более половины букв от конца названия, отличающихся на одну букву в середине названия).

Неверный набор значений термина легче обнаружить, если воспользоваться метрикой *объединение множеств значений одноименных терминов*.

Примером *дефекта* базы наблюдений урологического больного (одной версии ресурса, разрабатываемого в лаборатории интеллектуальных систем ИАПУ ДВО РАН и предназначенного для использования в подсистеме ведения историй болезни), на который можно "выйти" с помощью таких "вспомогательных" метрик, — значения "имеется" и "отсутствует" для характеристики "периодичность" наблюдаемого признака "рвота" (вместо традиционных для этой характеристики значений: однократно, 2—3 раза, многократно, постоянно, периодически).

Среди проблемно-независимых *потенциальных неудобств* базы знаний актуально свойство *многозначные термины*, одно из проявлений которого — *множество одноименных терминов с разной структурой* (определяется как число одноименных вершин в графе партономии). Это является неудобством при анализе/чтении и редактировании базы знаний, проверке полноты и корректности введенных знаний экспертом, при выборе подходящего термина для обозначения некоторой сущности при проектировании диалога с экспертом. В целях его выявления для рассматриваемого ре-

сурса строится *граф стандартной партономии*. Дуги графа принадлежат одному из трех типов — "состоит из" (для тех случаев, когда указывается множество однотипных частей), "включает часть" (когда каждая часть уникальна по своей структуре), "представлено альтернативой".

Множество одноименных терминов с разной структурой определяется как *множество* одноименных вершин в *графе партономии*, из которых выходят разные дуги или выходят дуги к разным вершинам. Для этого понадобится метрика *множество разных определений указанного термина*. И искомое *свойство-неудобство* будет определяться поэтапно.

Сначала определяется *множество имен терминов* ресурса как множество имен вершин в *графе партономии* этого ресурса.

Далее для каждого имени термина определяется метрика *множество разных определений указанного термина*.

И, наконец, свойство-неудобство *множество одноименных терминов с разной структурой* будет определяться как множество терминов, для которых мощность значений метрики *множество разных определений* больше единицы [9].

Пример. *Граф партономии*, построенный для *базы знаний заболеваний офтальмологии* (для одной версии базы знаний диагностической медицинской системы, разработанной также в лаборатории интеллектуальных систем ИАПУ ДВО РАН), содержит более сотни вершин, в том числе одноименных. Есть дуга "состоит из" от "качественные значения" к "значение", дуга "включает часть" от "качественные значения" к "реакция на воздействие события", дуга "представлено альтернативой" от "выбор типа значений" к "целые значения" и от "выбор типа значений" к "качественные значения".

Этап определения *множества имен терминов* ресурса: фрагмент полученного значения — {"характеристика", "выбор типа значений", "целые значения", "качественные значения", "вариант НР"...}.

Этап определения для каждого имени термина *множества разных определений указанного термина*: *множество разных определений* ("качественные значения") = {{значение (*)}, {вариант КП (*)},{вариант КП, КП, измененное воздействием событий}, {вариант (*)}, {вариант этиологии (*)},{вариант НР (*)}, {{вариант НР, реакция на воздействие события}}.

Этап определения свойства-неудобства *множество одноименных терминов с разной структурой*: фрагмент полученного значения — *множество одноименных терминов с разной структурой* = (..., "качественные значения", ...)

Заключение

Предложенный набор подзадач управления приложениями и подзадач мониторинга их информационных ресурсов, необходимый при разработке и управлении интеллектуальными программными системами, является основой для усовершенствования технологии разработки управляемых ИПС.

Предлагаемый метод мониторинга ресурсов на базе структурного подхода является основой для автоматизации тех компонентов инструментария, которые ответственны за управление решателями задач, за управление и согласованность документации и за качество используемых при проектировании моделей, декларативных компонентов самих разрабатываемых ИС, баз знаний и другой хранимой информации.

Описанный подход в настоящее время реализуется в виде экспериментального программно-информационного комплекса для проектирования, реализации и управления ИПС. Прототипная версия комплекса выполняется на двух языках программирования — Java и C++ (на C++ осуществляется эффективная аппаратно-зависимая реализация агентной платформы, а на Java — инструменты для высокоуровневого управления ИПС).

Работа выполнена при финансовой поддержке гранта РФФИ № 10-07-00089-а "Управление концептуальными метаонтологиями, онтологиями, знаниями и данными в интеллектуальных системах" и гранта ДВО РАН № 09-III-A-01-005 "Обеспечение качества интеллектуального программного обеспечения в процессе его проектирования на основе онтологий".

Список литературы

1. **Грибова В. В., Клещев А. С., Шалфеева Е. А.** Управление интеллектуальными системами // Известия РАН. Теории и системы управления. 2010. № 6. С. 122—137.

2. **Norvig P., Cohn D.** Adaptive software. URL: http://norvig.com/adapaper-pcai.html

3. **Средства Telelogic** для оптимизации процессов разработки // Byte. Июнь 2008. № 6 (116). URL: http://www.bytemag.ru/articles/detail.php?ID=12197

4. **Ayed D., Delanote D., Berbers Y.** MDD Approach for the Development of Context-Aware Applications // *Lecture Notes in Computer Science*. 2007. Berlin/Heidelberg: Springer. V. 4635. P. 15—28.

5. **Колесов А.** Российская действительность SOA: мнение поставщиков // PC Week/RE. 2008. № 20 (626). URL: http://www.pcweek.ru/themes/detail.php?ID=110251

6. **Пелц К.** Оркестровка и хореография Web-сервисов // Открытые системы (интернет-издание). 2004. URL: http://www.osp.ru/os/2004/11/184785/

7. **Calero C., Ruiz F., Piattini M. (eds).** Ontologies for Software Engineering and Software Technology. Springer Verlag, 2006. 339 p.

8. **Клещев А. С., Шалфеева Е. А.** Определение структурных свойств онтологий // Известия РАН. Теория и системы управления. 2008. № 2. С. 69—78.

9. **Shalfeeva E.** Monitoring of conceptual informational resources for intelligent software systems // Proc. First Russia and Pacific Conference on Computer Technology and Applications. 2010. P. 140—144.

10. **Шалфеева Е. А.** Методы оценивания структурных свойств онтологий, редактируемых и сохраняемых компьютерным банком знаний // Информатика и системы управления. 2007. № 2 (14). С. 132—142.

УДК 004.421.4; 004.052; 519.686; 514

Д. А. Орлов, канд. техн. наук, асс.,
Национальный исследовательский институт (Московский энергетический университет),
e-mail: orlovdmal@gmail.com

Выявление вычислительных аномалий в программных реализациях алгоритмов вычислительной геометрии*

В большинстве программных реализаций алгоритмов вычисления в действительных числах заменяются на вычисления с плавающей запятой. В большинстве случаев полученный результат не сильно отличается от истинного, и полученной точности достаточно для практических целей. Однако в некоторых случаях возможно получение результата, серьезно отличающегося от истинного. Подобные ситуации назовем вычислительными аномалиями.

Дается определение вычислительной аномалии, анализируются причины ее появления, предлагаются способы выявления вычислительных аномалий и тестирования существующих реализаций алгоритмов.

Ключевые слова: *вычислительная геометрия, числа с плавающей запятой, ошибки округления, вычислительные аномалии, тестирование программ*

Введение

Алгоритмы вычислительной геометрии широко используются для решения различных задач. Прежде всего это:

- компьютерная графика как статическая (CAD), так и реального времени (тренажерные системы, компьютерные игры);
- геоинформационные системы;
- компьютерное зрение;
- численные методы расчетов (в частности, построение расчетных сеток).

Особенностью рассматриваемого класса алгоритмов является их чувствительность как к ошибкам округления при задании входных данных, так и к вычислительным ошибкам внутри самого алгоритма. Вследствие этого, на выходе алгоритма можно получить результаты, качественно отличающиеся от истинных (пересекающиеся отрезки могут быть интерпретированы как непересекающиеся, выпуклый многоугольник может быть воспринят как невыпуклый). В некоторых случаях возможно даже зацикливание алгоритма [1]. Подобные ситуации назовем *вычислительными аномалиями.* Причиной возникновения ошибок округления является то, что в реализациях подобных алгоритмов числа, как правило, представлены в формате с плавающей запятой. Следовательно, арифметические операции над ними выполняются не точно.

На практике (для наблюдателя) в задачах трехмерной графики ошибки округления приводят к следующим нежелательным эффектам:

- некорректное (нереалистичное) поведение движущихся объектов виртуальной реальности;
- явные дефекты изображения (артефакты), такие, как:
  - разрывы в полигональных сетках трехмерных моделей и ландшафтов;

*Работа выполнена при поддержке РФФИ (грант № 11-07-00751-а).

- ♦ исчезновение или появление нежелательных объектов вследствие неправильного определения их видимости;
- ♦ дрожание анимации и другие [2];

• неверное определение наличия столкновений.

Пример артефактов изображения представлен на рис. 1 (см. третью сторону обложки). При визуализации изображения алгоритмом Z-буфера [3, 4] ошибки округления приводят к неверному определению видимости граней трехмерной модели.

Тем не менее, возможности возникновения вычислительных аномалий в процессе работы реализаций алгоритмов вычислительной геометрии уделяется недостаточно внимания. Кроме того, практически отсутствуют средства отладки, позволяющие определить возможность возникновения вычислительных аномалий в программных реализациях алгоритмов.

В данной статье предлагаются алгоритмы выявления вычислительных аномалий в реализациях алгоритмов вычислительной геометрии. Особое внимание уделено задаче обнаружения столкновений трехмерных объектов (*collision detection*) [5], поскольку она является неотъемлемой частью алгоритмов визуализации трехмерных сцен как статических (САПР), так и динамических (тренажерные системы, компьютерные игры). Предлагается метод выявления возможности возникновения вычислительных аномалий в программных библиотеках обнаружения столкновений.

1. Вычислительные аномалии

1.1. Последствия ошибок округления

В качестве примера неверного результата, полученного с использованием вычислений с плавающей запятой, приведем следующую задачу. Дана плоскость α (рис. 2), которая пересекает оси координат в следующих точках: $A(1009;0;0)$, $B(0;1013;0)$, $C(0;0;1019)$. Необходимо определить расположение точки D относительно данной плоскости, т.е. ориентации четверки точек. В данном случае, в зависимости от ориентации четверки точек, одна из точек находится или выше плоскости, или ниже, или на плоскости.

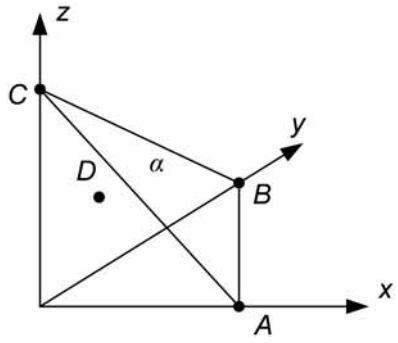

Рис. 2. Задача определения взаимного расположения точки и плоскости

Данная задача решается следующим образом: координаты точки D подставляются в уравнение плоскости α и анализируется знак полученного выражения. Если результат равен нулю, то точка D находится на плоскости α, если <0 — то ниже плоскости α, если >0 — то выше.

Для данной плоскости можно записать уравнение в отрезках:

$$\frac{x}{1009} + \frac{y}{1013} + \frac{z}{1019} - 1 = 0.$$

Это же уравнение можно переписать без использования деления:

$$1013 \cdot 1019x + 1009 \cdot 1019y + 1009 \cdot 1013z - \\ - 1009 \cdot 1013 \cdot 1019 = 0. \qquad (1)$$

Проверим точку $D(227; 802; -17)$. Данная точка находится ниже рассматриваемой плоскости. Точное значения выражения (1) для данной точки равно -1. Однако используя для вычисления значения выражения числа с плавающей запятой одинарной точности (фрагмент кода на языке *C*, используемый для вычислений, представлен на рис. 3), мы получим результат 6, что означает, что точка D была ошибочно воспринята как лежащая выше плоскости. Более того, используя фрагмент кода, представленный на рис.3, мы получим, что точки $A(1009; 0; 0)$, $B(0; 1013; 0)$, $C(0; 0; 1019)$ не принадлежат плоскости α. Обратите внимание, что максимальные абсолютные значения координат рассматриваемых точек в этом примере порядка 1000. Фрагмент кода, показанный на рис. 3, корректен, если считать арифметические операции идеальными, однако он использует числа с плавающей запятой одинарной точности (длина числа 32 бита). Следует также отметить, что при использовании чисел с плавающей запятой, результат будет зависеть от порядка выполнения операций.

1.2. Вычисления с плавающей запятой в стандарте IEEE754

Наиболее распространенным стандартом представления чисел с плавающей запятой в ЭВМ является стандарт IEEE 754-1985 [6]. В частности, он используется для представления вещественных чисел в x86-совместимых процессорах. Число с плавающей запя-

```
float res=-(float)a*(float)b*(float)c;
res+=(float)b*(float)c*(float)x+(float)a*(float)c*(float)y+(float)a*(float)b*(float)z;
```

Рис. 3. Фрагмент кода на языке C, дающий неверный результат

Таблица 1

Форматы чисел с плавающей запятой, определенные стандартом IEEE 754—1985

Формат	Длина, бит	Номер бита, задающего знак	Разряды экспоненты, ее длина, значение ее смещения	Разряды мантиссы, ее длина	Соответствующий тип языка C
Короткий	32	31	30—23, 8 бит, 127	22—0, 23 бит	float
Длинный	64	63	62—52, 11 бит, 1023	51—0, 52 бит	double
Расширенный	80	79	78—64, 15 бит, 16383	63—0, 64 бит	long double

той состоит из мантиссы q и экспоненты p. Само число задается формулой

$$x = q \cdot 2^p.$$

В стандарте IEEE 754-2008 [7] определены также числа с десятичной экспонентой, в дальнейшем будем рассматривать только числа с двоичной экспонентой, как наиболее часто используемые. В стандарте IEEE 754—1985 используются три формата представления чисел с плавающей запятой: короткий, длинный и расширенный. Сведения об этих форматах приведены в табл. 1.

Если результат арифметической операции не помещается в разрядную сетку, происходит его округление. Стандарт IEEE 754 содержит несколько режимов округления: с недостатком (к $+\infty$), с избытком (к $-\infty$), к ближайшему представимому числу, к 0. В x86-совместимых процессорах, как правило, используется округление до ближайшего представимого числа. Кроме того, в стандарте IEEE 754 предусмотрено исключение и флаг, которые сигнализируют о произошедшем округлении результата. В сопроцессорах x86-совместимых процессоров этому флагу соответствует флаг PE [8].

Вычисления с плавающей запятой имеют следующие недостатки:

- точно можно представить только рациональные числа, знаменателем которых является целая степень числа 2, в общем случае любое действительное число представляется с погрешностью;
- нарушаются свойства арифметических операций (например, свойство ассоциативности операции сложения);
- существует явление существенной потери значимости при вычитании двух близких чисел;
- наряду с обычным переполнением имеется антипереполнение (обнуление мантиссы при округлении результата, близкого к нулю).

1.3. Подходы к снижению влияния ошибок округления на результаты алгоритмов

Основными подходами к снижению влияния ошибок округления на результаты алгоритмов вычислительной геометрии, являются:

1) увеличение разрядности чисел;

2) проверка чисел на равенство с введением интервала нечувствительности;

3) изменение порядка арифметических операций;

4) предварительная обработка геометрических данных;

5) отдельная обработка вырожденных и близких к ним случаев.

Наиболее очевидным подходом является увеличение разрядной сетки чисел с плавающей запятой. В некоторых случаях этот подход может быть эффектным, однако он не устраняет вычислительные ошибки полностью. Оригинальная реализация этого способа, ориентированная на 3D-графику реального времени, представлена в работе [2].

Следующий подход состоит в замене проверок равенства двух чисел с плавающей запятой проверками на их достаточную близость, т.е. проверками того, что модуль их разности меньше достаточно малого числа ε. Подобный прием используется при проверке на равенство нулю результата некоторого выражения (например, при проверке нахождения трех точек на одной прямой). Однако в этом случае необходимо достаточно точно оценить ε, чтобы правильно полученные различные значения не принять за ноль.

Третий подход — внесение изменений в порядок операций над числами. Например, желательно избегать сложений и вычитаний чисел, сильно различающихся по абсолютной величине.

Четвертый подход состоит в преобразовании исходных данных. Например, при работе с полигональными сетками желательно разбивать многоугольники на треугольники, так как три точки всегда лежат в одной плоскости, а при перемещении многоугольника из-за ошибок округления его вершины могут перестать находиться в одной плоскости.

Пятый подход состоит в разбиении геометрической задачи на несколько случаев и использовании для каждого из них своего алгоритма решения. Желательно отдельно рассматривать более простые частные случаи, вырожденные случаи, а также случаи, наиболее подверженные влиянию ошибок округления (например, случай пересечения "почти параллельных" прямых) [9].

Большинство перечисленных подходов имеет следующие недостатки: чрезмерное усложнение алгоритма, отсутствие универсальности, кроме того, все эти под-

ходы не гарантируют отсутствие вычислительных аномалий. Поэтому представляются интересными как проверка имеющейся реализации алгоритма на подверженность вычислительным аномалиям, так и выявление недостоверных результатов в процессе вычислений.

2. Вычислительные аномалии

2.1. Формализация понятия "Вычислительная аномалия"

В данном разделе проводится исследование механизма возникновения вычислительных аномалий, вызванных ошибками округления, т.е. в качестве причины возникновения вычислительных аномалий будем рассматривать только ошибки округления. Рассмотрим некий алгоритм и его реализацию, использующую вычисления с плавающей запятой.

Определение. Под *реализацией алгоритма* будем понимать машинный код алгоритма, выполняющийся на конкретной вычислительной системе.

Очевидно что результат реализации алгоритма может не совпадать с результатом абстрактного алгоритма. Так как в данном исследовании рассматриваются вычислительные аномалии, источником которых являются только ошибки округления, сделаем следующее допущение.

Допущение 1. Будем считать, что различие между алгоритмом и его реализацией состоит только в замене действительных чисел на один из типов данных, доступных для данной вычислительной системы.

Из этого следует, что:

- множество наборов входных данных реализации алгоритма является подмножеством множества наборов входных данных исходного алгоритма;
- операции над числами в реализации алгоритма, использующей вычисления с плавающей запятой, могут проводиться неточно (с ошибкой округления), т.е. в некоторых случаях результат выполнения реализации алгоритма будет отличаться от истинного результата алгоритма.

Исследуем причины возникновения вычислительных аномалий в реализациях алгоритмов, использующих вычисления с плавающей запятой.

Пусть в исследуемом алгоритме необходимо вычислить значение $y = f(X)$. Если $f(x)$ непрерывна в точке X и в достаточно большой ее окрестности, то ошибки округления оказывают незначительное влияние на результат вычисления $f(X)$ (рис. 4, *а*). Если $f(x)$ имеет точки разрыва вблизи точки X, то ошибки округления могут привести к результату, не имеющему ничего общего с истинным (рис. 4, *б*), т.е., к возникновению вычислительной аномалии. На рис. 4: X — истинное значение аргумента функции, X^* — вычисленное значение аргумента функции, искаженное под влиянием ошибок округления.

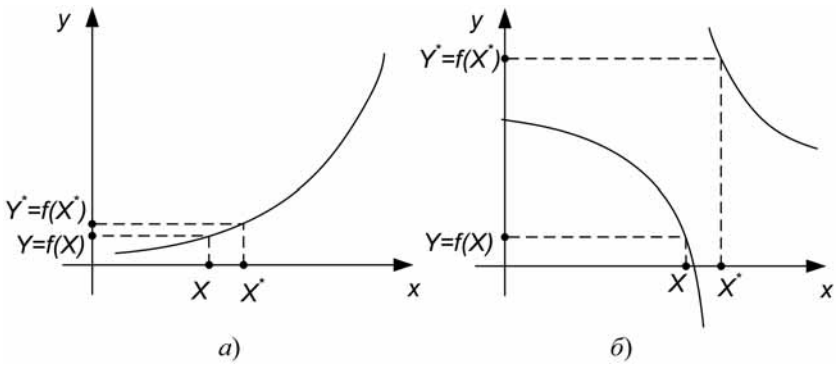

Рис. 4. Влияние ошибок округления в случаях непрерывной функции (*а*) и функции, имеющей разрыв (*б*)

Определение. *Качественно неверным результатом вычисления функции* $f(x)$, имеющей хотя бы одну точку разрыва, назовем значение $f(X^*)$ для вычисленного значения аргумента X^*, которое из-за ошибок округления принадлежит иному, чем истинное значение аргумента X, интервалу непрерывности функции $f(x)$. Другими словами, когда отрезок $\lfloor \min(X^*, X); \max(X^*, X) \rfloor$ содержит хотя бы один разрыв функции $f(x)$.

В алгоритмах вычислительной геометрии из функций, имеющих разрывы, чаще всего используются: $\mathrm{sgn}(x)$, т.е. функция определения знака числа, и функция $y = \frac{1}{x}$. Операцию сравнения двух чисел a и b можно представить как $\mathrm{sgn}(a-b)$, следовательно, на ее результат ошибки округления также могут оказать значительное влияние.

Приведем пример получения качественно неверного результата. Точное значение выражения

$$\left(\frac{30029}{30031} - \frac{15014}{15015}\right)^{-1} \qquad (2)$$

составляет 450915465. Однако фрагмент кода на языке C, приведенный на рис. 5, даст результат −18127660, резко отличающийся от истинного.

Подобная разница между истинным значением выражения и результатом выполнения программы возникает вследствие неточного вычисления разности дробей. При этом абсолютная погрешность вычисления знаменателя выражения (2) невелика, но точное и вычисленное с помощью фрагмента кода, приведенного

```
float d=30031.f/30029.f;
float res=1/(1/d-15014.f/15015.f);
```

Рис. 5. Фрагмент кода на языке C, дающего неверный результат из-за ошибок округления

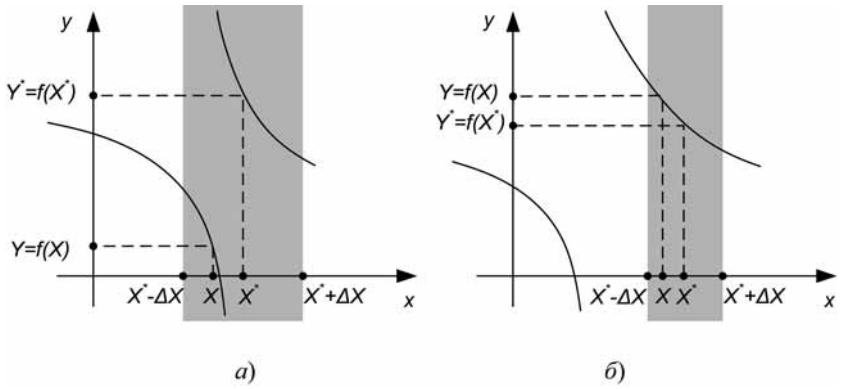

Рис. 6. Недостоверный (*а*) и достоверный (*б*) результаты вычисления функции, имеющей разрыв

на рис. 5, значения знаменателя имеют противоположные знаки, что приводит к существенному различию значений их обратных величин. В этом примере в качестве $f(x)$ выступает функция $y = \frac{1}{x}$.

Определить факт получения качественно неверного результата можно лишь зная истинный результат выражения. Для обнаружения возможности возникновения качественно неверного результата в случае, когда точный результат неизвестен (например в процессе вычисления выражения), введем следующие определения.

Пусть вычисленное значение аргумента X^* имеет верхнюю оценку ошибки округления ΔX, т.е., что выполняется следующее неравенство:

$$X^* - \Delta X \leqslant X \leqslant X^* + \Delta X, \qquad (3)$$

(т.е. истинное значение аргумента содержится в интервале $[X^* - \Delta X; X^* + \Delta X]$). Тогда можно ввести понятие "недостоверный результат вычисления функции".

Определение. *Недостоверным результатом вычисления функции* $f(x)$ назовем значение $f(X^*)$ для вычисленного значения аргумента X^*, имеющего верхнюю оценку ошибки округления $\Delta X \geqslant 0$ (удовлетворяющую условию (3)), такую, что в интервал $[X^* - \Delta X; X^* + \Delta X]$ входит хотя бы один разрыв функции $f(x)$.

Результат, представленный на рис. 6, *а*, является недостоверным, так как разрыв функции лежит внутри полученного интервала (интервал выделен серым). Результат, представленный на рис. 6, *б*, является достоверным, так как полученный интервал (выделен серым) не содержит разрывов функции $f(x)$.

Очевидно, что если $f(X^*)$ является качественно неверным результатом, то это значение будет являться недостоверным результатом (так как ΔX — оценка погрешности сверху). Поскольку, если X — истинное значение аргумента, то между X и X^* находится хотя бы одна из точек разрыва $f(X^*)$. Поэтому справедливо следующее утверждение.

Утверждение 1. Отсутствие недостоверных результатов в процессе вычисления алгоритма означает отсутствие качественно неверных результатов.

В данной работе целесообразно сформулировать следующее ограничение.

Ограничение. Будем рассматривать алгоритмы, которые содержат только следующие операции и функции, выполняемые над действительными числами: сложение, вычитание, умножение, деление, определение знака числа — $\mathrm{sgn}(x)$, сравнение двух чисел.

Данное ограничение сужает множество исследуемых алгоритмов, однако ему удовлетворяют алгоритмы решения наиболее распространенных задач вычислительной геометрии: построение результатов булевых операций над многогранниками, обнаружение столкновений (если объекты заданы многогранниками или сферами), построение выпуклой оболочки, построение триангуляции множества точек.

Утверждение 2. Если выходными данными алгоритма являются значения, принадлежащие конечным множествам, то отсутствие качественно неверных результатов в реализации алгоритма гарантирует совпадение ее результатов с истинными результатами алгоритма.

С учетом ограничения на функции и операции, используемые в алгоритме, данное утверждение можно доказать следующим образом. Если выходная переменная рассматриваемого алгоритма может принимать конечное множество значений, то ее значение вычисляется путем выполнения ряда сравнений. Если результат каждого сравнения верный, то будет получено истинное значение такой переменной.

Выходные данные большинства алгоритмов вычислительной геометрии содержат значения, принадлежащие конечным множествам. Поэтому, исходя из утверждения 2, сформулируем следующее допущение.

Допущение 2. В качестве причины возникновения вычислительных аномалий будем рассматривать только получение качественно неверного результата вычисления функции.

Исходя из сформулированных допущений, дадим определение понятию "вычислительная аномалия" для реализаций алгоритмов вычислительной геометрии.

Определение. *Вычислительная аномалия* (для реализаций алгоритмов вычислительной геометрии) — это возникновение одной из следующих ситуаций:

- получение качественно неверного результата в выходных данных реализации алгоритма;
- зацикливание реализации алгоритма.

Определение. Реализацию алгоритма будем считать *подверженной вычислительным аномалиям*, если хотя бы для одного набора входных данных, точно представ-

ленных в ее входном формате, возникает вычислительная аномалия.

Отметим, что при введенных допущениях зацикливание реализации алгоритма является следствием получения качественно неверного результата в промежуточных данных алгоритма.

Утверждение 3. При введенных допущениях отсутствие недостоверных результатов в реализации алгоритма гарантирует отсутствие вычислительных аномалий.

Данное утверждение является следствием определений вычислительной аномалии и недостоверного результата.

Сформулированный в данном разделе механизм возникновения вычислительных аномалий в алгоритмах вычислительной геометрии далее будет использован для разработки алгоритмов исключения вычислительных аномалий и исследования реализаций алгоритмов вычислительной геометрии на подверженность вычислительным аномалиям.

2.2. Методы поиска вычислительных аномалий

Методы поиска вычислительных аномалий можно разделить на два класса: тестирующие алгоритм как "черный ящик" и использующие исходные тексты или машинные коды алгоритма. Первый класс методов позволяет ответить на вопрос: есть ли возможность возникновения вычислительной аномалии в данной реализации алгоритма. Наличие вычислительной аномалии проверяется путем сравнения эталонных выходных данных с выходными данными исследуемой реализации алгоритма. Второй класс методов позволяет обнаружить факт и место возникновения вычислительной аномалии на произвольном наборе входных данных. В следующих разделах будут описаны конкретные методы обоих классов.

При тестировании функций как "черных ящиков" важно проверить сохранение тех свойств их идеальных прототипов, которые явно или неявно использует реализуемый алгоритм.

2.3. Поиск вычислительных аномалий в процессе выполнения программы

Как было показано в разд. 2.1, возможность возникновения вычислительной аномалии следует из факта получения недостоверного результата. Поэтому необходимо контролировать достоверность результата. Один из способов контроля — оценка ошибки округления, присутствующей в результате [10—12].

Разрабатываемый метод оценки достоверности результата вычислений с плавающей запятой основан на определении оценки ошибки округления одновременно с вычислением результата выражения. Очевидно, что в этом случае для получения подобной оценки достаточно заменить в исходном коде алгоритма тип данных с плавающей запятой на тип, представляющий "число с оценкой ошибки округления". Затем, если вычисляемая функция имеет разрыв и оценка погрешности аргумента функции такова, что в полученный интервал попадает хотя бы один из разрывов функции, то результат признается недостоверным. Подобный тип вычислений называют вычислениями с автовалидацией.

Определение. *Вычисления с автовалидацией* — это набор алгоритмов представления чисел, сравнения чисел, и выполнения операций над этими числами, который позволяет в процессе вычисления определить факт получения недостоверного результата.

Примером вычислений с автовалидацией могут служить интервальные вычисления [13, 14].

Существуют два основных способа представления числового интервала:

1. Представление своими границами — для интервала $[c; d]$, $c \leqslant d$ в памяти ЭВМ запоминаются именно его границы. Вычисления, оперирующие подобными объектами, называют интервальными.

2. Представление числом и погрешностью — это представление интервала в виде пары чисел $a \pm \Delta a$, где a — середина интервала, Δa — погрешность, с которой известно число a, $\Delta a \geqslant 0$.

Оба этих представления эквивалентны, поскольку $a \pm \Delta a$ задает интервал $[a - \Delta a; a + \Delta a]$, а интервал $[c; d]$ можно задать как $\frac{d+c}{2} \pm \frac{d-c}{2}$.

Чтобы длина интервала соответствовала верхней оценке погрешности вычислений, все операции над интервалами должны обладать следующим свойством: интервал, представляющий результат операции над исходными интервалами, должен содержать в себе множество результатов соответствующей операции над числами из этих интервалов. Аналогично, результаты операций над числами с погрешностью должны задавать интервал, содержащий в себе множество результатов соответствующей операции над числами из интервалов, представленных каждым из аргументов.

Это свойство может быть сформулировано иначе: если число не принадлежит результирующему интервалу операции, то оно не является истинным результатом операции.

Рассмотрим способы выполнения арифметических операций над интервалами, представленными своими границами. Предположим, что ошибки округления при вычислении границ интервалов отсутствуют. Тогда получим следующие интервалы для суммы, разности, произведения, обратной и противоположной величин, а также для частного интервалов (табл. 2).

Если число x представлено интервалом $[c; d]$ содержащим 0, то, согласно определению, данному в разд. 2.1, результат выражения $\frac{1}{x}$ будет признан недостоверным.

Таблица 2

Результаты арифметических операций над интервалами

Операция	Результирующий интервал
$[c_1; d_1] + [c_2; d_2]$	$[c_1 + c_2; d_1 + d_2]$
$[c_1; d_1] - [c_2; d_2]$	$[c_1 - d_2; d_1 - c_2]$
$[c_1; d_1] \cdot [c_2; d_2]$	$[\min(c_1d_1, c_1d_2, c_2d_1, c_2d_2); \max(c_1d_1, c_1d_2, c_2d_1, c_2d_2)]$
$[c; d]^{-1}$, $cd > 0$	$[1/d; 1/c]$
$-[c; d]$	$[-d; -c]$
$[c_1; d_1]/[c_2; d_2]$, $c_2d_2 > 0$	$[c_1; d_1] \cdot [c_2; d_2]^{-1}$

Таблица 3

Результаты операций для интервальных вычислений при округлении результатов

Операция	Результирующий интервал
$[c_1; d_1] + [c_2; d_2]$	$[c_1 \underline{+} c_2; d_1 \overline{+} d_2]$
$[c_1; d_1] - [c_2; d_2]$	$[c_1 \underline{-} d_2; d_1 \overline{-} c_2]$
$[c_1; d_1] \cdot [c_2; d_2]$	$[\min(\underline{c_1d_1}, \underline{c_1d_2}, \underline{c_2d_1}, \underline{c_2d_2}); \max(\overline{c_1d_1}, \overline{c_1d_2}, \overline{c_2d_1}, \overline{c_2d_2})]$
$[c; d]^{-1}$, $cd > 0$	$\left[\underline{\frac{1}{d}}; \overline{\frac{1}{c}}\right]$
$-[c; d]$	$[-d; -c]$
$[c_1; d_1]/[c_2; d_2]$, $c_2d_2 > 0$	$[c_1; d_1] \cdot [c_2; d_2]^{-1}$

Если границы интервалов представлены числами с плавающей запятой, то при их вычислении неизбежны округления. Следовательно, возможны случаи, в которых истинное значение не будет принадлежать результирующему интервалу. Для исключения таких случаев необходимо принудительно задавать режимы округления. Значение нижней границы интервала должно быть округлено с недостатком, а верхней — с избытком. В этом случае результирующий интервал будет содержать в себе истинный результат проведенной операции. Исходя из этого, модифицируем формулы, представленные в табл. 2. Введем следующие обозначения: результат операции, подчеркнутой снизу, округляется с недостатком, результат операции, подчеркнутой сверху, округляется с избытком. Тогда результаты операций, представленные в табл. 2 примут следующий вид (табл. 3).

Рассмотрим способы выполнения арифметических операций над интервалами, представленными как число и погрешность. В качестве оценки выберем максимальную теоретическую оценку ошибки округления.

Предположим, что ошибки округления при вычислении результатов отсутствуют. Тогда получим следующие интервалы для суммы, разности, произведения и обратной величины (табл. 4).

Отметим, что если $a \leqslant \Delta a$, то результат вычисления $\frac{1}{a}$ будет признан недостоверным.

Если число и оценка его погрешности представлены числами с плавающей запятой, то при их вычислении неизбежны округления. Следовательно, и здесь возможны случаи, в которых истинное значение не будет принадлежать результирующему интервалу. Для исключения таких случаев, при вычислении оценки погрешности также необходимо принудительно задавать режимы округления. Исходя из этого, модифицируем формулы, представленные в табл. 4.

Принцип модификации формул для оценки погрешности операций состоит в следующем: необходимо установить режимы округления таким образом, чтобы получить наибольшее значение оценки погрешности. Кроме того, необходимо проверить, имело ли место округление результата операции. Если да, то к оценке погрешности необходимо добавить (округляя результат с избытком) $d/2$, где d — вес младшего разряда мантиссы результата соответствующей операции. В FPU x86-совместимых процессоров наличие округления можно определить по флагу PE состояния со-

Таблица 4

Оценки максимальной погрешности для различных арифметических операций над интервалами, представленными как число и погрешность

Операция	Оценка погрешности
$a + b$	$\Delta a + \Delta b$
$a - b$	$\Delta a + \Delta b$
$a \cdot b$	$\lvert a\rvert\Delta b + \lvert b\rvert\Delta a + \Delta a\Delta b$
$\frac{1}{a}$, $a > \Delta a$	$\Delta a/(\lvert a\rvert(\lvert a\rvert - \Delta a))$

Таблица 5

Оценки максимальной погрешности для различных арифметических операций над интервалами, представленными как число и погрешность, при наличии ошибок округления

Операция	Оценка погрешности
$a + b$	$\Delta a \overline{+} \Delta b \overline{+} d$
$a - b$	$\Delta a \overline{+} \Delta b \overline{+} d$
$a \cdot b$	$\lvert a\rvert \overline{\cdot} \Delta b \overline{+} \lvert b\rvert \overline{\cdot} \Delta a \overline{+} \Delta a \overline{\cdot} \Delta b \overline{+} d$
$\frac{1}{a}$, $a \geqslant \Delta a$	$\Delta a \overline{/} (\lvert a\rvert \underline{:} (\lvert a\rvert \underline{=} \Delta a)) \overline{+} d$

процессора [8]. Тогда результаты операций, представленные в табл. 4, примут следующий вид (табл. 5).

Отметим, что универсальные программные интерфейсы для задания режимов округления и определения факта получения неточного результата в настоящее время отсутствуют, что ограничивает возможности написания кросс-платформенных реализаций приведенных типов вычислений [14].

Оценка погрешности результата анализируется только перед выполнением операций, для которых возможно получение недостоверного результата. К таким операциям относятся: вычисление функций, имеющих разрывы; проверка условий ветвления (если две ветви имеют существенное различие в обработке). Перед вычислением функции $f(x)$, имеющей разрыв в точке X, необходимо проверить условие:

$$|a - X| \leqslant \Delta a, \tag{4}$$

где a — число в формате с плавающей запятой, передаваемое на вход $f(x)$, Δa — оценка погрешности для a. Если условие (4) выполнено, результат вычисления функции $f(x)$ считается недостоверным. В этом случае выражение должно быть вычислено повторно с использованием вычисления большей точности или вычисления с исключением ошибок округления. Аналогично, при сравнении двух чисел a и b, имеющих оценки погрешности Δa и Δb соответственно, выполнение условия

$$|a - b| \leqslant |\Delta a + \Delta b|, \tag{5}$$

означает недостоверность результата и необходимость повторного вычисления хотя бы одного из чисел a и b.

Для алгоритмов вычислительной геометрии, оперирующих прямыми или отрезками, используются функции $\mathrm{sgn}(x)$ и $1/x$. Для этих функций условие (5) превращается в:

$$|a| \leqslant \Delta a.$$

Разработанный алгоритм оценки ошибок округления реализован в программной библиотеке, написанной на языке C++.

Словесное описание алгоритма следующее:

1. Присвоение оценки ошибки округления каждому входному числу. Такая оценка должна быть равной 0 в случае, если входные данные вычислены и представлены точно, или $d/2$ (где d — вес младшего разряда мантиссы числа) в случае, если при преобразовании в формат с плавающей запятой произошло округление. На этом шаге можно задать любое значение ошибки округления в исходных данных.

2. Для всех исследуемых арифметических операций над числами с плавающей запятой в алгоритме повторить шаги с 3 по 6.

3. Если функция (или операция) имеет особые точки (разрыв), например, $1/x$, перейти к шагу 4, иначе перейти к шагу 6.

4. Если в результате текущей операции может быть получен недостоверный результат, то проверить возможность возникновения вычислительной аномалии на данной операции, используя формулы (4) или (5).

5. Если проверяемое условие ((4) или (5), в зависимости от текущей операции) выполнено, то результат считается недостоверным и текущая ситуация требует особой обработки (например, может быть вызвано исключение). Иначе — переход к шагу 6.

6. Вычисление результата операции, а также оценка погрешности вычисленного результата. Вычисление оценки погрешности происходит с использованием формул, представленных в табл. 2. Кроме того, к полученной таким образом оценке погрешности следует добавить $d/2$ в случае, если при вычислении результата операции произошло округление.

Если получен недостоверный результат, то существует возможность возникновения вычислительной аномалии. Обработка таких результатов может быть проведена вручную. В общем случае вычисления должны быть повторены с использованием более точного представления с плавающей запятой или вычисления с исключением ошибок округления.

Подобный алгоритм может быть полезен при отладке программ для определения возможности возникновения вычислительных аномалий. Недостаток алгоритма следующий: так как используется максимальное теоретически возможное значение погрешности (оценка сверху), то при оценке погрешности результата программы, проводящей достаточно длинную цепочку вычислений (например, достаточное большое количество итераций при решении задачи численным методом), оценка погрешности результата может оказаться сильно завышенной, поэтому может резко повыситься число ложных срабатываний (алгоритм может выдавать сигнал о возможности получения недостоверного результата почти для всех входных значений). Однако для небольших отлаживаемых участков кода алгоритм будет достаточно точно детектировать возможность возникновения аномалий. Подобные участки часто возникают в предикатах вычислительной геометрии.

Следует сказать, что результат выполнения операций над числами с плавающей запятой зависит от процессора, на котором выполняется программа, от используемой операционной системы и от настроек компилятора. Не во всех комбинациях этих параметров вычисления осуществляются строго по стандарту IEEE 754 [13]. В некоторых случаях возможна повышенная ошибка округления, отсутствие некоторых режимов округления или флага неточного результата. Данная библиотека разработана для x86-совместимых процессоров (так как использует управление режима-

ми работы сопроцессора через команды ассемблера). Тестирование проводилось для Windows XP SP3 (версия среды разработки VisualStudio 2005 SP1) и для OpenSUSE Linux 11.2 (версия компилятора GCC 4).

2.4. Поиск вычислительных аномалий в существующих реализациях алгоритмов

Предлагаемый метод состоит в вычислении результатов реализации алгоритма для специально сгенерированных наборов данных и сравнении полученных результатов с эталонными результатами. Это дает возможность тестировать реализации алгоритмов, не имея их исходных текстов. Единственная информация, которая необходима — это формат чисел с плавающей запятой, использующийся в исследуемой реализации. Предполагается, что исследуемая реализация алгоритма использует вычисления с плавающей запятой (формат float или double стандарта IEEE 754). Кроме того, алгоритм должен удовлетворять ограничениям, введенным в разд. 2.1.

Метод включает в себя следующие шаги.

Шаг 1. Выбор частного случая задачи, которую решает тестируемая реализация алгоритма (реализация I_T). Выбранный частный случай назовем модельной задачей. Более того, он должен быть подзадачей, возникающей при решении задачи для общего случая. На этом же шаге выбирается диапазон входных переменных.

Шаг 2. Создание двух тестовых реализаций для выбранного частного случая: использующую вычисления с плавающей запятой (реализация I_f) и использующую вычисления с исключением ошибок округления (реализация I_R). I_f необходима для оценки ошибок округления, таким образом может быть получена косвенная оценка ошибок округления, возникающих в реализации I_T. Реализация I_R обладает следующим свойством: для каждого входного набора данных, точно представимого в формате входных данных I_f, результат I_R совпадает с результатом абстрактного алгоритма. Следовательно, с помощью I_R могут быть получены эталонные результаты.

Шаг 3. Оценка ошибок округления, возникающих в I_f.

Шаг 4. Генерация входных наборов данных, для которых реализация I_f возвращает недостоверный результат (используя при этом ошибку округления, полученную на шаге 3).

Шаг 5. Вычисление и сравнение результатов реализаций I_T, I_f, I_R для наборов данных, сгенерированных на шаге 4. Если хотя бы для одного из сгенерированных наборов данных результат реализации I_T отличается от результата реализации I_R, тогда реализация I_T считается подверженной вычислительным аномалиям.

Предложенный метод упрощает нахождение входных данных, для которых тестируемая реализация возвращает неверный результат. Однако совпадение результатов I_T и результатов I_R для сгенерированных данных не означает, что реализация I_T не подвержена вычислительным аномалиям.

Применим разработанный метод к существующим реализациям алгоритмов обнаружения столкновений.

3. Эксперимент по тестированию библиотек обнаружения столкновений

3.1. Задача обнаружения столкновений

Задача обнаружения столкновений (*collision detection*) [5] состоит в обнаружении факта пересечения или касания геометрических тел. Методы обнаружения столкновений трехмерных объектов применяются в системах визуализации, алгоритмах геометрического моделирования. Основные их применения: компьютерные игры, тренажерные системы, САПР.

Существуют два подхода к решению задачи обнаружения столкновений.

1. Приближенный: объекты аппроксимируются телами, проверка на пересечение с которыми выполняется значительно быстрее. В качестве подобных тел выступают сферы, параллелепипеды, цилиндры, а также их объединения, организованные в деревья ограничивающих объемов [5]. Библиотеки, реализующие данный подход, жертвуют точностью в угоду скорости, при этом они подходят для задач, где возможность возникновения вычислительной аномалии не является столь критичной (например, для компьютерных игр).

2. Точный: на пересечение проверяются геометрические тела, представляющие объекты. Однако приемы, описанные в предыдущем пункте, могут использоваться для ускорения вычислений. Подобные библиотеки могут применяться как в графике реального времени, так и в САПР.

В дальнейшем будем рассматривать библиотеки, реализующие подход 2, так как они должны давать точный ответ на вопрос о наличии столкновения. Как правило, модели трехмерных объектов представляются в виде многогранников. Поэтому задача обнаружения столкновений сводится к задаче пересечения многогранников [5].

На данный момент существует множество реализаций алгоритмов решения данной задачи, оформленных в виде программных библиотек: SOLID [15], ColDet [16], OPCODE [17], COLLIDE [18]. Во многих библиотеках числа представляются в формате с плавающей запятой, следовательно, результаты арифметических операций над ними содержат ошибки округления.

Геометрические тела могут задаваться следующим образом:

- как результат операций над более простыми телами (конструктивная геометрия);

- как полигональная сетка;
- как неупорядоченный набор многоугольников (*polygon soup*);
- аналитически;
- как результат пересечения нескольких полупространств.

Наиболее распространенными способами для библиотек, реализующих точный подход, являются представление объекта полигональной сеткой или неупорядоченным набором многоугольников. В этом случае для обнаружения факта пересечения двух тел важной становится подзадача обнаружения взаимного расположения точки и плоскости.

3.2. Особенности генерации тестовых данных

Применим метод, предложенный в разд. 2.4, для тестирования библиотек обнаружения столкновений.

Для библиотек обнаружения столкновений в качестве модельной выберем задачу определения взаимного расположения точки и плоскости, так как эта задача является основной подзадачей для определения факта пересечения двух многогранников, кроме того, алгоритм ее решения достаточно прост. Модельную задачу будем решать при следующих ограничениях:

1. Плоскость задается уравнением в отрезках $\frac{x}{a} + \frac{y}{b} + \frac{z}{c} - 1 = 0$ (где a, b, c — натуральные числа).

2. Ограничимся точками с целыми координатами, для которых ($0 \leqslant x \leqslant 2a$; $0 \leqslant y \leqslant 2b$; $-c \leqslant z \leqslant c$).

Тогда взаимное расположение точки (x, y, z) и плоскости определяется знаком выражения:

$$bcx + acy + abz - abc. \qquad (6)$$

Поскольку для вычисления значения выражения (6) используются числа с плавающей запятой, оно будет вычислено с погрешностью. Однако для большинства точек знак выражения (6) будет определен верно, что в данном случае означает отсутствие вычислительной аномалии.

После установки этих ограничений можно оценить максимальную величину ошибки округления, возникающей при вычислении данного выражения. Для выбранной области трехмерного пространства (в случае, если используется режим округления к ближайшему представимому числу):

$$\varepsilon = 3\mathrm{val}(abc),$$

где $\mathrm{val}(x)$ вес младшего разряда мантиссы числа с плавающей запятой, ближайшего к x.

Доказательство:

$$\Delta(bcx + acy + abz - abc) \leqslant \\ \leqslant \Delta(bcx) + \Delta(acy) + \Delta(abz) + \Delta(abc),$$

где $\Delta(x)$ — верхняя оценка ошибки округления для x. Поскольку область пространства, в которой будут генерироваться точки, ограничена (см. выше), верно следующее неравенство:

$$\Delta(bcx) + \Delta(acy) + \Delta(abz) + \Delta(abc) \leqslant \\ \leqslant \Delta(2bcx) + \Delta(2acb) + \Delta(abc) + \Delta(abc).$$

Благодаря особенностям формата чисел с плавающей запятой стандарта IEEE 754, нормализованные числа x и $2x$ имеют одинаковые мантиссы, поэтому $\Delta(2bca) = 2\Delta(bca)$. Тогда:

$$\Delta(bcx + acy + abz - abc) \leqslant 6\Delta(abc).$$

Если abc имеет точное представление во внутреннем формате FPU (для архитектуры x86 это 80-битный формат) [8] и используется режим округления к ближайшему представимому числу, тогда $\Delta(abc) = \mathrm{val}(abc)/2$. Иными словами:

$$\Delta(bcx + acy + abz - abc) \leqslant 3\mathrm{val}(abc).$$

Выбор чисел a, b и c необходимо провести таким образом, чтобы их произведение представлялось в формате с плавающей запятой неточно. Также, НОД(a, b) и НОД(b, c) должны быть равны 1, так как в этом случае сгенерированные точки будут ближе к плоскости.

Из допущений, сформулированных в разд. 2.1, следует, что знак выражения (6) может быть определен неверно только в случае, если значение этого выражения меньше (по абсолютной величине) ошибки округления с которой оно получено. Следовательно, необходимо сгенерировать набор точек, для которых будет выполняться неравенство:

$$|bcx + acy + abz - abc| \leqslant \varepsilon. \qquad (7)$$

Точки с целыми координатами, удовлетворяющие неравенству (7), генерировать полным перебором затруднительно, так как в этом случае имеет место задача сложности порядка $O(n^2)$, (где n — наибольшее из чисел a и b), что для значений координат уже порядка 10000 может оказаться неприемлемым. Поэтому необходим другой способ генерации точек.

Поскольку рассматриваются только точки с целыми координатами, выражение (6) может принимать только целые значения. Таким образом, в данном случае выражение (7) эквивалентно следующей совокупности Диофантовых уравнений:

$$\left[\begin{array}{l} |bcx + acy + abz - abc| =]\varepsilon[\\ |bcx + acy + abz - abc| =]\varepsilon[-1 \\ \qquad\qquad ... \\ |bcx + acy + abz - abc| = 0 \\ \qquad\qquad ... \\ |bcx + acy + abz - abc| = -]\varepsilon[+1 \\ |bcx + acy + abz - abc| = -]\varepsilon[. \end{array}\right.$$

Поэтому задача генерации данных сводится к задаче решения данной совокупности Диофантовых уравнений.

3.3 Результаты эксперимента

Для тестирования выбраны библиотеки solid-3.5.6 и ColDet 1.2, так как они реализуют точный подход к обнаружению столкновений, свободно доступны и хорошо документированы.

Согласно разд. 2 разработанной методики, в ходе эксперимента необходимо проверить следующие реализации алгоритма решения модельной задачи:

- вычисление выражения (6) с использованием целых чисел изменяемой разрядности. В данном случае использование целых чисел изменяемой разрядности корректно, так как рассматриваются только точки с целыми координатами, а использование этого типа чисел гарантирует отсутствие округления и переполнения. Поэтому данную реализацию алгоритма будем считать эталонной;
- вычисление выражения (6) с использованием чисел в формате *float*;
- вычисление выражения (6) с использованием чисел в формате *double*;
- реализация алгоритма с использованием библиотеки solid-3.5.6;
- реализация алгоритма с использованием библиотеки ColDet 1.2.

Для тестирования выбрано (согласно требованиям, приведенным в разд. 3.2) следующее уравнение плоскости:

$$\frac{x}{1000003} + \frac{y}{1000033} + \frac{z}{1000037} - 1 = 0.$$

Числа в уравнении подобраны таким образом, что их произведение $>2^{57}$, а значит оно представляется в формате с плавающей запятой двойной точности (*double*) неточно. Выбранные числа взаимно-простые, поскольку важно, чтобы сгенерированные точки с целыми координатами не лежали бы на плоскости, но при этом имели бы как можно меньшее расстояние до нее.

Для проверки приведенных выше реализаций на подверженность вычислительным аномалиям проведена генерация точек в следующей области пространства:

$$\begin{cases} 0 \leqslant x \leqslant 2000006 \\ 0 \leqslant y \leqslant 2000066 \\ -1000037 \leqslant z \leqslant 1000037. \end{cases}$$

Наибольшая оценка погрешности вычисления выражения (6) в выбранной области пространства составила 384. Число точек, для которых результат недостоверен — 3081. Далее показано число точек, для которых каждая из исследованных реализаций алгоритма дала неверный результат. Генерация данных проведена исходя из оценки ошибок округления типа *double*.

Реализация	Число неверных результатов
Тестовая, вычисления двойной точности	951
Тестовая, вычисления одинарной точности	1545
solid-3.5.6	2304
ColDet 1.2	1533

Эксперимент был проведен еще раз, теперь с использованием чисел с плавающей запятой одинарной точности. Для тестирования выбрано следующее уравнение плоскости:

$$\frac{x}{1009} + \frac{y}{1013} + \frac{z}{1019} - 1 = 0.$$

Для проверки приведенных выше реализаций на подверженность вычислительным аномалиям проведена генерация точек в области пространства:

$$\begin{cases} 0 \leqslant x \leqslant 2018 \\ 0 \leqslant y \leqslant 2026 \\ -1019 \leqslant z \leqslant 1019. \end{cases}$$

Оценка погрешности вычисления выражения (6) составила 192. Число точек, для которых результат недостоверен — 1545. Далее показано число точек, для которых каждая из исследованных реализаций алгоритма дала неверный результат. Генерация данных проведена исходя из оценки ошибок округления для типа *float*.

Реализация	Число неверных результатов
Тестовая, вычисления двойной точности	0
Тестовая, вычисления одинарной точности	37
solid-3.5.6	1419
ColDet 1.2	400

Проведенный эксперимент по определению подверженности библиотек обнаружения столкновений (*collision detection*) вычислительным аномалиям показал, что:

- все исследованные реализации алгоритма обнаружения столкновений, использующие вычисления с плавающей запятой, подвержены вычислительным аномалиям, вне зависимости от разрядности мантиссы;
- для задачи обнаружения столкновений разработан алгоритм генерации данных, при обработке которых в реализациях алгоритма обнаружения столкновений, использующих вычисления с плавающей запятой, велика возможность возникновения вычислительных аномалий;

• существует возможность намеренной генерации данных в целях получения вычислительной аномалии в заданной реализации алгоритма, причем для генерации данных подобного типа достаточно лишь знать разрядность чисел с плавающей запятой, используемых в алгоритме.

Заключение

В статье была рассмотрена проблема влияния ошибок округления на результаты алгоритмов. Формализованы понятия вычислительная аномалия и недостоверный результат. Вычислительные аномалии рассматриваются как последствия недостоверных результатов вычислительных операций. Приведены примеры вычислительных аномалий (данные и участки кода, позволяющие их воспроизвести). Также разработан метод поиска вычислительных аномалий в алгоритмах, для которых имеются исходные коды. Частью данного метода является разработанный автором алгоритм оценки погрешности результатов операций над числами с плавающей запятой. Данный алгоритм реализован программно.

Разработан метод проверки имеющихся реализаций алгоритмов вычислительной геометрии на подверженность вычислительным аномалиям. С помощью данного метода исследованы библиотеки обнаружения столкновений. Полученные результаты говорят о подверженности данных библиотек вычислительным аномалиям. Кроме того, ошибки в результатах разных библиотек имеют сходный характер.

Разработанные методы и алгоритмы могут найти применение при отладке и разработке программного обеспечения, использующего вычисления с плавающей запятой. В особенности, для разработки и отладки реализаций алгоритмов вычислительной геометрии.

Список литературы

1. **Kettner L., Mehlhorn K., Pion S., Schirra S., Yap C.** Classroom Examples of Robustness Problems in Geometric Computations // ESA, LNCS. 2004. Vol. 3221. P. 702—713. URL: http://www.mpi-inf.mpg.de/~kettner/pub/nonrobust_cgta_06.pdf

2. **Freese P.** Solving Accuracy Problems in Large World Coordinates // Game Programming Gems 4, Charles River Media. 2004. P. 157—170.

3. **Ву М., Нейдер Д., Девис Т., Шрайнер Д**. OpenGL. Официальное руководство программиста: пер. с англ. СПб: ООО "ДиаСофтЮп", 2002. 592 с.

4. **Шикин А. В., Боресков А. В.** Компьютерная графика. Полигональные модели. М.: ДИАЛОГ-МИФИ, 2001. 464 с.

5. **Ericson C.** Real-Time Collision Detection. The Morgan Kaufmann, 2005.

6. **Стандарт** IEEE 754-1985.

7. **Стандарт** IEEE 754-2008.

8. **Зубков С. В.** Assembler для DOS, Windows и Unix. М.: ДМК, 1999. 640 с.

9. **Piegl L.** Knowledge-guided Computation for Robust CAD // Computer-Aided Design & Applications. 2005. V. 2. N 5. P. 685—695.

10. **Orlov D.** Application of computations with calculation error exclusion for computation geometry algorithms // Proc. of International Conference on dependability of Computer Systems DepCOS — RELCOMEX 2009. Brunow, Poland, 30 June — 2 July 2009. Los Alamitos: IEEE Computer Society. 2009. P. 290—295.

11. **Melhorn K., Näher S.** LEDA A Platform for Combinatorial and Geometric Computing. URL: http://www.mpi-inf.mpg.de/%7Emehlhorn/LEDAbook.html

12. **Дзегеленок И. И., Харитонов В. Ю., Орлов Д. А.** Вычислительные аспекты построения распределенных систем виртуальной реальности // Вестник Московского Энергетического института. 2008. № 5. С. 27—32.

13. **Collange S., Flórez J., Defour D.** A GPU interval library based on Boost.Interval. URL: http://hal.inria.fr/docs/00/26/36/70/PDF/interval_gpu_fpl_v2.pdf

14. **Библиотека** интервальных вычислений Boost.Interval. URL: http://www.boost.org/doc/libs/1_37_0/libs/numeric/interval/doc/interval.htm

15. **Библиотека** SOLID. URL: http://www.dtecta.com/

16. **Библиотека** ColDet. URL: http://www.photoneffect.com/coldet/

17. **Библиотека** OPCODE. URL: http://www.codercorner.com/Opcode.htm

18. **Библиотека** COLLIDE. URL: http://www.cs.unc.edu/~geom/collide/index.shtml

ИНФОРМАЦИЯ ДЛЯ ЧИТАТЕЛЕЙ

Статья **В.А.Васенина, С.А. Афонина, Д.Д.Голомазова** "Использование семантических технологий для обнаружения Грид-ресурсов", опубликованная в № 7 за 2011 г., выполнена при финансовой поддержке гос. контракта 07.514.11.4116 Министерства образования и науки РФ

УДК 004.7

В. А. Васенин, д-р физ.-мат. наук, проф., зав. лабораторией,
А. А. Иткес, канд. физ.-мат. наук, мл. науч. сотр.,
Ф. М. Пучков, канд. физ.-мат. наук, стар. науч. сотр.,
К. А. Шапченко, канд. физ.-мат. наук, стар. науч. сотр,
НИИ механики МГУ имени М. В. Ломоносова,
e-mail: vasenin@msu.ru,

Обеспечение информационной безопасности в распределенных системах на основе технологий Grid и Cloud Computing: традиционные средства защиты и вопросы асимметрии доверия*

Рассматриваются типовые подходы к обеспечению информационной безопасности в распределенных системах, построенных на основе технологий Grid Computing и Cloud Computing. Выделяется задача реализации взаимного доверия поставщика и потребителя Grid- и Cloud-услуг, обсуждаются подходы к ее решению.

Ключевые слова: *информационная безопасность, распределенные системы, Grid Computing, Cloud Computing, асимметрия доверия, Trusted Computing, Trusted Platform Module, гомоморфное шифрование*

Использование высокопроизводительных Grid- и Cloud-систем в современных условиях развития информационных технологий мотивировано как с организационно-технической, так и с экономической точек зрения. Ресурсов для решения многих вычислительно емких и практически важных задач в рамках одной организации или с использованием одного вычислительного комплекса оказывается недостаточно. Вместе с тем, решение подобных задач становится возможным при объединении ресурсов (включая вычислительные узлы, системы хранения информации, наборы данных, библиотеки методов решения задач) в рамках кооперации нескольких заинтересованных организаций. Кроме того, некоторые задачи оказывается экономически выгоднее решать, воспользовавшись услугами, которые предоставляются в рамках некоторой внешней вычислительной инфраструктуры, несмотря на то, что ресурсы для их решения на собственной технической базе в отдельной организации есть.

* Работа выполнена при частичной финансовой поддержке гос. контракта 07.514.11.4116 Министерства образования и науки РФ.

Принимая во внимание отмеченные выше доводы, следует иметь в виду, что объединение организаций в Grid-среду и использование Cloud-услуг ставят ряд дополнительных задач по обеспечению информационной безопасности ресурсов, которые при этом используются [1, 2]. Далее в кратком изложении перечислим основные соображения относительно подходов к решению этих задач. С одной стороны, обеспечение информационной безопасности может быть необходимо для реализации действий над данными с ограниченным режимом доступа в распределенной системе, если такие задачи перед этой системой ставятся. С другой стороны, обеспечение информационной безопасности может осуществляться в целях снижения ущерба от нарушения обязательств в отношениях "поставщик—потребитель", которые реализуются в рамках распределенной системы, а именно — для повышения уровня взаимного доверия между поставщиком и потребителем услуг. С третьей стороны, методы обеспечения информационной безопасности могут использоваться для реализации дополнительных гарантий внутренней целостности и надежности данных, которые циркулируют в распределенной систе-

ме, а также гарантий корректности работы сервисов такой системы.

Для рассматриваемых в настоящей работе распределенных систем (Grid- и Cloud-систем) выделим и систематизируем их характеристики как объектов защиты. Определим общие характеристики рассматриваемых систем, включая их назначение, программную архитектуру, а также интерпретируем понятие информационной безопасности для таких систем.

Grid- и Cloud-системы как объект защиты. Рассмотрим две распространенные в настоящее время схемы построения высокопроизводительных распределенных систем при совместном участии в этом процессе нескольких заинтересованных сторон. В классической постановке задач Grid Computing [3, 4] это схема "виртуальной организации", а при использовании технологий Cloud Computing — схема поставки набора однородных вычислительных ресурсов в режиме "по запросу".

В традиционном представлении систем, построенных на основе технологий Grid Computing [5], элементами верхнего архитектурного уровня этих систем являются так называемые "виртуальные организации". В рамках таких организаций несколько заинтересованных сторон объединяют принадлежащие им вычислительные средства и активы данных (далее — ресурсы) в единую высокопроизводительную распределенную систему для решения задач, представляющих для них совместный интерес. В качестве объединяемых ресурсов, как правило, используются вычислительные сервисы, которые реализуются с помощью высокопроизводительных установок, имеющихся в распоряжении участников "виртуальной организации". Это обстоятельство свидетельствует об относительно небольшом количестве объединяемых ресурсов. Возможности по объему ресурсов, предоставляемых одной из участвующих сторон, определяются, как правило, числом выделяемых ей узлов высокопроизводительных вычислительных установок. Отдельные ресурсы в получившейся Grid-системе не всегда оказываются взаимозаменяемыми. В силу отличий объединяемых в рамках "виртуальной организации" компонентов, такая система как объединение многих предоставленных ресурсов, как правило, будет гетерогенной по программно-аппаратным платформам в ее составе, по правилам хранения и доступа к данным отдельных ее участников.

Типовые вычислительные системы, построенные по схеме Cloud Computing, в общем случае организованы по следующим принципам. Поставщик услуг Cloud Computing предоставляет каждому из своих клиентов (пользователей) ресурс распределенного хранения данных и набор вычислительных ресурсов. Эти ресурсы являются взаимозаменяемыми для пользователя и предоставляют единый программный интерфейс доступа к управлению вычислительными процессами. Ресурсы предоставляются пользователю в режиме "по запросу". В силу взаимозаменяемости ресурсов и при условии их достаточно большого количества поставщик может обеспечить и значительный объем ресурсов, предоставляемых отдельному пользователю. Заметим однако, что те взаимозаменяемые вычислительные ресурсы, которые удается аккумулировать в необходимом количестве, как правило, каждый по-отдельности обладают низкой производительностью, а сетевая инфраструктура их поддержки может иметь относительно высокие задержки при передаче данных.

Интерпретация термина "информационная безопасность" в применении к Grid- и Cloud-системам. Одним из первых шагов процесса по созданию организационной системы управления информационной безопасностью согласно международным стандартам ISO/IEC 27001:2005 и ISO/IEC 27002:2005 (ранняя версия которого положена в основу стандарта Российской Федерации ГОСТ Р 17799-2005) является определение и интерпретация понятия "информационная безопасность" в применении к защищаемой системе. Отметим, что в силу отсутствия необходимости в единой точке организационного управления в рассматриваемых системах, применение положений указанных стандартов в явном виде может оказаться не совсем корректным. Вместе с тем, по мнению авторов, многие положения этих стандартов применительно к Grid- и Cloud-системам не теряют своей силы. В рамках настоящей работы будем использовать традиционное определение общего понятия информационной безопасности как "защищенности информации и поддерживающей инфраструктуры от случайных или преднамеренных воздействий естественного или искусственного характера, которые могут нанести неприемлемый ущерб субъектам информационных отношений" [6]. Для интерпретации этого понятия в применении к информационно-вычислительным системам на основе Grid- и Cloud-технологий рассматриваются следующие его составляющие: информационные отношения; субъекты таких отношений; информация; поддерживающая инфраструктура; возможный ущерб.

Информация и информационные отношения определяются высокоуровневыми процессами взаимодействия субъектов в распределенной системе. Для обозначения таких процессов, как правило, используются термины: бизнес-процессы, композитные приложения, *scientific workflow* и другие.

Для субъектов информационных отношений, которые потенциально могут взаимодействовать в распределенной системе, можно выделить несколько ролей, в том числе пересекающихся, когда один субъект может выступать в нескольких ролях:

- производители и поставщики средств вычислительной техники;
- владельцы средств вычислительной техники и поставщики услуг по предоставлению средств вычислительной техники;
- владельцы сетевой инфраструктуры и поставщики услуг связи между средствами вычислительной техники;
- владельцы и поставщики информации;
- владельцы и поставщики сервисов среднего слоя;
- владельцы и поставщики информационно-вычислительных сервисов;

- конечные пользователи (клиенты информационно-вычислительных сервисов).

Отмеченная в определении информационной безопасности поддерживающая инфраструктура в широком смысле представляет собой компоненты распределенной системы, необходимые для реализации информационных отношений (от аппаратного обеспечения до средств среднего слоя и средств обеспечения безопасности).

Недопустимый по требованиям информационной безопасности уровень ущерба для субъектов информационных отношений в Grid-среде зависит от подлежащих решению с ее помощью задач. Такая зависимость может быть наглядно продемонстрирована на примере следующих типовых задач, выполняемых в Grid-среде: учебные трудоемкие вычислительные задачи (предположительно низкий уровень ущерба); задачи обработки данных со строгим режимом доступа, например, персональных данных (предположительно высокий уровень ущерба даже при нарушении конфиденциальности либо целостности малого количества таких данных); задачи предоставления информационно-вычислительных услуг (предположительно низкий или средний уровень ущерба). Следует учесть, что для некоторых задач одни аспекты безопасности более существенны по сравнению с другими. Например, если система используется для научных исследований, результаты которых публикуются в открытом доступе, нарушение целостности информации может оказаться более нежелательным, чем нарушение конфиденциальности.

Типовые подходы к разработке программной архитектуры комплекса средств обеспечения безопасности. Рассмотрим подходы к созданию программной архитектуры средств обеспечения безопасности элементов Grid- и Cloud-систем на основе известных в этой области типовых решений. В их числе:

- *Globus Security Infrastructure* (GSI) — средства обеспечения безопасности в пакете Globus Toolkit;
- рекомендации по средствам безопасности в архитектурном стиле REST;
- традиционные подходы к организации средств обеспечения безопасности в Cloud-системах.

В GSI [7—9] выделяются четыре функции, связанные с безопасностью: защита отдельных сообщений (реализация требований спецификаций WS-Security и WS-SecureConversation); аутентификация (причем отмечается необходимость взаимной аутентификации, когда не только клиент аутентифицирует себя перед Grid-сервисом, но и наоборот); делегирование привилегий (на основе прокси-сертификатов в X.509); авторизация. Защиту каналов связи от несанкционированного доступа в рамках GSI предлагается реализовать с помощью протокола TLS с использованием сертификатов в формате X.509 и с поддержкой прокси-сертификатов. Отметим, что перечисленные четыре функции не предназначены для защиты от атак вида "отказ в обслуживании". Кроме того, заметим, что в GSI рассматриваются два вида компонентов Grid-систем, а именно — веб-сервисы и более ранние компоненты, не основанные на унифицированном наборе технологий реализации сервисо-ориентированной архитектуры в составе Globus Toolkit (так называемых "WS-*технологий"). В обоих видах компонентов используются одни и те же механизмы аутентификации, однако компоненты, отличные от веб-сервисных, как правило, используют свои собственные протоколы для обеспечения безопасности передачи данных.

В отличие от GSI архитектурный стиль REST не определяет заранее набора технологий и средств обеспечения безопасности. Как следствие, разработчики сервисов на основе REST должны реализовать собственные механизмы и средства защиты. Как правило, выделяются две составляющих в обеспечении безопасности — аутентификация (например, на основе технологий X.509, OAuth, OpenID, а также упрощенной парольной аутентификации) и разграничение доступа. Предлагается использовать программный механизм SSL как способ защиты каналов передачи данных от несанкционированного доступа. Однако по технологиям обеспечения конфиденциальности и целостности отдельных передаваемых сообщений в традиционных источниках по архитектурному стилю REST рекомендаций не дается.

Основная идея реализации архитектуры средств безопасности в рамках стиля REST заключается в том, что механизмы безопасности формируются в виде сервисов-посредников (прокси-сервисов). В качестве таких посредников следует отметить: сервисы для принятия решения о доступе к другому сервису; сервисы для балансировки нагрузки и их возможное использование для организации противодействия атакам на отказ в обслуживании; сервисы для сбора данных аудита. Механизмы логического разграничения доступа в рамках стиля REST также предлагается реализовывать как сервисы-посредники. Все запросы к сервису, к которому осуществляется доступ, в рамках такого подхода проходят через отдельный сервис принятия решения о доступе. Следует отметить, что с помощью сервисов-посредников, выполняющих функции балансировки нагрузки и функции разграничения доступа, могут быть частично уменьшены негативные последствия от атак на отказ в обслуживании через исчерпывание вычислительных и/или коммуникационных ресурсов.

Для типовых Cloud-систем характерна централизация управления, например, под контролем единственного владельца. Как следствие, внутри такой системы действует единый набор правил безопасности, а задача интеграции политик безопасности не ставится. Элементы типовых Cloud-систем однородны по своим свойствам (или приводятся к таковым). Этот факт позволяет избежать значительных локальных отличий в механизмах безопасности. Обычно выделяются следующие основные задачи обеспечения безопасности, которые стоят перед поставщиком услуг Cloud-системы: аутентификация пользователей Cloud-системы; изоляция одних пользователей от других и изоляция

пользователей от инфраструктуры Cloud-системы; обеспечение отказоустойчивости системы. Задача изоляции пользователей друг от друга и от инфраструктуры, как правило, решается с помощью средств виртуализации. Отметим, что такие средства также используются в составе системы для достижения более высокого уровня загруженности вычислительных ресурсов. Задача повышения отказоустойчивости традиционно решается дублированием инфраструктуры, включая, например, вычислительные узлы Cloud-системы и узлы хранения.

Обеспечение безопасного выполнения пользовательского программного кода в распределенной системе. Перечислим традиционно используемые технологии безопасного выполнения пользовательских программ на стороне поставщика вычислительных средств. Широко распространено применение механизмов изоляции в операционных системах (ОС) и средств виртуализации, включая средства изоляции на уровне процессов операционной системы, средства паравиртуализации, средства запуска одной ОС как процесса в другой, средства виртуализации аппаратного обеспечения.

Альтернативным способом безопасного выполнения программы является предъявление специальных ограничивающих требований к программам, предназначенным для выполнения в распределенной среде. К их числу относятся, например, использование специализированных языков программирования, библиотек и правил по оформлению программ, за счет чего реализуются дополнительные ограничения на код, выполняемый в распределенный среде. Кроме того, среда выполнения таких программ сама по себе зачастую может ограничивать выполнение и изолировать одни программы от других (например, реализуя модель "песочница"). Отметим, что одной из традиционных форм распространения программ в распределенной среде для высокопроизводительных вычислений является передача исходного кода с последующей компиляцией для целевой платформы (в гетерогенной среде таких платформ несколько). В таком случае работа по проверке ограничений на программы может быть возложена на сервис компиляции, включая функции по проверке кода и встраиванию в него дополнительных (но несложных в силу общей трудоемкости автоматических операций по анализу и изменению семантики исходного кода) ограничений.

Использование аппаратных криптографических средств в целях обеспечения доверия к поставщику вычислительных ресурсов. Традиционное использование средств обеспечения безопасности в Grid- и Cloud-системах нацелено на защиту поставщиков услуг от потребителей (пользователей). Подобная асимметричность может привести к тому, что недобросовестный поставщик каких-либо услуг в рабочем процессе в распределенной системе может, получив доступ к данным пользователя, нанести ему ущерб. В таких условиях актуальны задачи об ограничении действий поставщика услуг и о проверке такого ограничения потребителем. Заметим, что здесь под услугами, в первую очередь, понимаются услуги по предоставлению некоторых вычислительных ресурсов для выполнения на них задач потребителя.

Одним из подходов к решению отмеченной выше задачи является так называемая технология доверенных вычислений (*Trusted computing*, далее этот термин обозначает только одну специализированную технологию и не является обобщением для терминов доверие и вычисления) на основе доверенных аппаратных криптографических средств, которая продвигается организацией Trusted Computing Group [10]. Основная идея применения технологии доверенных вычислений в задаче ограничения действий поставщика вычислительных услуг следующая. Предполагается, что аппаратная платформа, предоставляемая поставщиком, является доверенной, в том числе в ней есть доверенный центральный процессор, доверенная память, доверенный набор базовых системных микроконтроллеров, доверенная BIOS и специальное доверенное криптографическое устройство, называемое *Trusted Platform Module* (TPM). В процессе загрузки аппаратной платформы доверенная BIOS подсчитывает контрольную сумму загрузчика операционной системы (ОС) и помещает ее в TPM (одной из функций TPM является защищенное хранение контрольных сумм в так называемых регистрах конфигурации платформы (*Platform Configuration Registers*, PCR). Загрузчик ОС, в свою очередь, подсчитывает контрольную сумму ядра ОС и помещает ее в TPM, расширяя уже хранимую в PCR контрольную сумму загрузчика. Аналогичную операцию проделывает ядро ОС, подсчитывая контрольную сумму выполняемых приложений. Затем, в процессе функционирования системы от TPM может быть получено значение регистра конфигурации платформы, подписанное уникальным закрытым ключом TPM, и передано потребителю для проверки. Зная открытый ключ для проверки подписи TPM (например, от производителя TPM) и доверяя реализации аппаратных компонентов, потребитель удостоверяется в подлинности переданной ему контрольной суммы. Сравнив контрольную сумму с некоторым известным ему эталоном, потребитель убеждается, что он взаимодействует именно с той программно-аппаратной платформой, о составе которой он договаривался ранее с поставщиком.

Заметим, что, как правило, набор программного обеспечения, необходимый потребителям, может значительно различаться, а поставщику удобно (например, в целях уменьшения числа возможных контрольных сумм) иметь единообразный набор программного обеспечения на своих вычислительных узлах. Это обстоятельство привело к расширению технологии доверенных вычислений до технологии доверенной виртуализации. В ее рамках поставщик предоставляет единую программно-аппаратную платформу для запуска виртуальных окружений. В каждом таком окружении выполняются образы операционных систем, предоставляемые пользователями. Прототип реализации

такой идеи представлен в работе [11], где предложена архитектура программно-аппаратной платформы для доверенной виртуализации и протокол взаимодействия для проведения удаленной аттестации платформы пользователем.

Проведение вычислений над данными в зашифрованном виде. Ранее уже отмечались проблемные вопросы, возникающие в современных Grid- и Cloud-системах, связанные с асимметричностью в обеспечении безопасности поставщиков и потребителей (пользователей) услуг рассматриваемой отрасли. В этой связи все большую актуальность приобретает задача обеспечения конфиденциальности персональных данных пользователя, имея в виду возможные угрозы со стороны недобросовестных поставщиков вычислительной техники. В предыдущем подразделе рассмотрен один из подходов к решению данной задачи, основанный на использовании специальных доверенных аппаратных криптографических модулей. В данном пункте будет рассмотрен принципиально новый подход, который основан на концепции вычислений над зашифрованными данными и схеме гомоморфного шифрования.

Гомоморфным называется шифрование, которое позволяет осуществлять определенные математические действия с открытым текстом путем проведения (в общем случае других) операций с зашифрованным текстом. Многие известные алгоритмы шифрования (например, RSA, ElGamal, гаммирование) являются гомоморфными относительно одной операции — умножения или сложения. Однако в 2009 г. Крэйгом Гэнтри впервые была предложена полностью гомоморфная схема шифрования (относительно умножения и сложения одновременно) [12]. Тогда, используя только лишь указанные две операции (умножение и сложение), легко выразить все остальные математические операции, включая побитовые операции и операции сравнения.

С использованием гомоморфного шифрования, схема вычислений над зашифрованными данными будет выглядеть следующим образом (см. рисунок). Предположим, что потребителю нужно вычислить некоторую функцию F над данными X. При этом он желает сохранить результат вычисления в тайне от поставщика высокопроизводительной вычислительной системы. Пользователь с помощью специального алгоритма шифрования E и секретного ключа k шифрует данные X. Затем передает пару $(F, E_k(X))$ поставщику вычислительной системы. Поставщик, приняв эту пару, вычисляет значение функции F на данных $E_k(X)$ и передает обратно пользователю значение $F(E_k(X))$. В силу свойства гомоморфности алгоритма шифрования по отношению к любой вычислимой функции F, будет справедливо равенство $F(E_k(X)) = E_k(F(X))$. Тогда потребитель, используя алгоритм расшифрования D, зная секретный ключ k, получает требуемый результат, а именно — $D_k(E_k(F(X))) = F(X)$. Таким образом, поставщик проводит вычисления над зашифрованными данными, не требуя ключа для их расшифрования. Тем самым обеспечивается конфиденциальность данных потребителя, в том числе по отношению к поставщику вычислительной системы.

Отметим, однако, что в настоящее время подобная схема может быть реализована лишь в теории. Данное обстоятельство связано с тем, что при увеличении размера пространства секретных ключей, быстро растет и число операций, которые необходимы для вычисления над зашифрованными данными, а именно, при использовании n-битных секретных ключей сложность вычисления значения $F(E_k(X))$ равна $O(n^{3,5}N)$, где N — сложность вычисления значения $F(X)$. Например, при использовании 100-битных ключей, время, которое потребуется для вычислений над зашифрованными данными будет в 10 000 000 раз больше того, которое необходимо для обычных вычислений. Таким образом, в будущем подход, основанный на концеп-

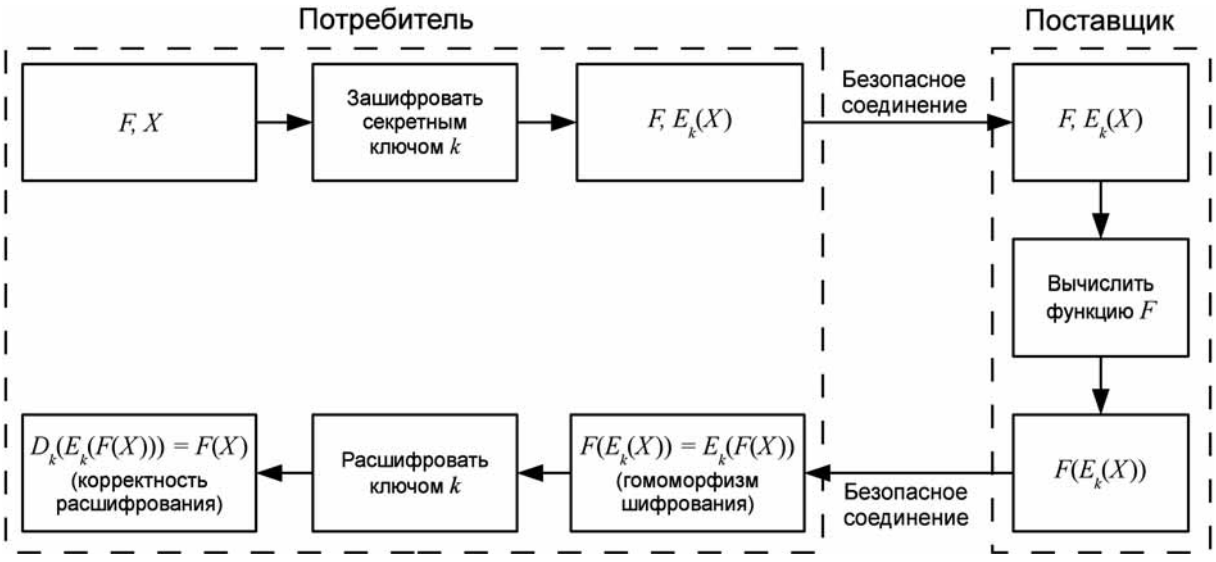

Схема вычислений с использованием гомоморфного шифрования

ции вычислений над зашифрованными данными, может быть перспективным. Однако, и это следует отметить, в настоящее время его применение нецелесообразно.

Методы формальной верификации в задачах обоснования выполнения требований к распределенным системам. В современных Grid- и Cloud-системах, предназначенных для выполнения важных с практической точки зрения задач, требования, предъявляемые к их результатам, как правило, формулируются в терминах выполнения некоторых процессов (в других источниках — бизнес-процессов, композитных приложений), которые должны быть реализованы в таких распределенных системах. Каждый процесс обычно состоит из нескольких шагов, использующих ресурсы одного или нескольких участников взаимодействия. Необходимость использования подобных процессов, в первую очередь, обусловлена неоднородностью компонентов системы по аппаратной и программной составляющим. В подобных условиях технология бизнес-процессов представляется хорошим унифицированным способом разрешения вопросов, возникающих при интеграции разнородных приложений в распределенной, гетерогенной среде. Однако для построения распределенной информационной системы (ИС) с высоким уровнем защищенности необходимо самостоятельно осуществлять контроль за выполнением глобальной политики информационной безопасности. С этих позиций верификация бизнес-процессов является одним из перспективных средств поддержания высокого уровня защищенности распределенной информационной системы.

Вопрос принятия решения о допустимости бизнес-процессов возникает, когда проводится процедура интеграции в ИС нового информационного запроса (нового вида информационной услуги). Инициатором интеграции и разработчиком бизнес-процесса в ряде случаев может являться не только администратор ИС, но и пользователь. Следовательно, перед выполнением указанной процедуры администратору безопасности необходимо удостоверится, что бизнес-процесс не представляет угрозы получения несанкционированного доступа к ресурсам информационной системы или атаки на отказ в обслуживании.

Кроме того, корректность каждого такого процесса является необходимым условием для гарантированного получения тех результатов, для которых была создана высокопроизводительная распределенная система. Корректность здесь понимается в широком смысле этого понятия, включая корректность построения процесса, корректность результата при выполнении заданных предусловий, корректность по отношению к требованиям по безопасности и корректность с позиции возможного неопределенного поведения. Такая корректность должна быть как проверена до запуска самого процесса, так и подтверждена при контроле состояния его функционирования в динамике.

Для решения рассматриваемой задачи представляется целесообразным применить методы верификации бизнес-процессов. Для этого администратор безопасности должен составить список правил, которым должны следовать все процессы, интегрированные в информационную систему. В качестве примера разумного набора правил можно указать следующее: перед каждым обращением к сервисам внутренних информационных систем бизнес-процесс должен обратиться к специальным сервисам безопасности, осуществляющим проверку корректности такого доступа в рамках существующей политики безопасности. Отметим, что задача проверки корректности процессов в распределенной системе усложняется в силу необходимости учета особенностей заданной конфигурации для элементов операционной среды, для компонентов среднего слоя и для средств обеспечения безопасности.

Выводы. На основании представленных выше распространенных и уже прошедших апробацию решений, а также подходов к обеспечению информационной безопасности в Grid- и Cloud-системах, можно сделать следующие выводы.

Основные вопросы обеспечения безопасности, которые рассматриваются в применении к Grid-системам, — аутентификация и авторизация, а также смежные понятия, например, распределение аутентификационной информации, делегирование привилегий через посредников. Кроме того, выделяется (однако реже, чем аутентификация и авторизация) задача обеспечения безопасности транспортного уровня передачи данных. Другие сервисы безопасности — например, обнаружение вторжений, мониторинг состояния работоспособности, обеспечение отказоустойчивости — упоминаются, как правило, не в контексте Grid-систем как таковых, а при их рассмотрении как одного из частных случаев распределенной системы произвольного вида.

Существующие рекомендации по использованию технологий обеспечения безопасности не позволяют сделать выбор технологий в зависимости от уровня допустимого (согласно заданным требованиям) ущерба. Как следствие, при таком использовании технологии обеспечения безопасности предоставляют только некоторую дополнительную внутреннюю целостность в распределенной системе без каких-либо явно поставленных целей, связанных со снижением ущерба от деструктивных информационных воздействий. В архитектуре средств обеспечения безопасности выделяются только технологические аспекты реализации, но не вопросы проверки корректного использования выбранных средств и технологий.

Кроме того, из представленной интерпретации понятия информационная безопасность в применении к Grid-системам может быть сделан вывод, что необходимо защищать как поставщиков Grid- и Cloud-услуг, так и потребителей. Однако технологии обеспечения безопасности применяются асимметрично: больше защищается поставщик от потребителя, чем потребитель от поставщика. Поставщик может контролировать выполнение процессов потребителя, что позволяет нейтрализовать или снизить возможный ущерб

от несанкционированных действий последнего. Таким образом, у поставщика услуг есть возможность защиты от злоумышленных действий со стороны потребителя, например, через разграничение доступа, механизмы изоляции, средства активного аудита, средства повышения отказоустойчивости при высокой нагрузке. Потребитель, напротив, как правило, доверяет поставщику услуг полностью. Со своей стороны, он не обладает средствами контроля за принадлежащей ему информацией, которая обрабатывается средствами вычислительной техники поставщика. В том числе, потребитель открывает поставщику свои данные (включая исполняемые программы), передача которых другим лицам может принести прямой или косвенный ущерб потребителю Grid- и Cloud-услуг.

Заметим, что традиционно используемые методы аутентификации позволяют удостовериться в такой ситуации, что участники информационных отношений являются именно теми, кем они представляются, но не позволяют гарантировать безопасного обслуживания. Таким образом, неконтролируемые вычисления над данными в открытом виде — один из источников риска для конечного пользователя. На направлении снижения такого риска перспективными представляются исследования в области технологии *Trusted Computing* и в области вычислений над зашифрованными данными. Однако в последней отмеченной области исследований — вычислениях над зашифрованными данными (т.е. данными, открытое представление которых получает только обладатель ключа, при условии стойкости используемой криптосистемы) — в настоящее время не получено результатов, позволяющих эффективно использовать такие вычисления на практике. Это подчеркивает необходимость наличия дополнительного независимого арбитра для обоснования гарантий поставщика перед потребителем, как например, предлагается делать в рамках технологии *Trusted Computing* с помощью доверенных унифицированных аппаратных криптографических средств.

Значительное влияние на выбор методологии и средств обеспечения информационной безопасности в Grid- и Cloud-системах оказывают особенности архитектуры таких систем. В качестве основных факторов, влияющих на выбор, следует отметить децентрализованную структуру таких систем, большое число компонентов в их составе, гетерогенность по используемым техническим средствам, различия в установленных требованиях и правилах безопасности. Указанные особенности приводят к усложнению организации инфраструктуры распространения аутентификационных данных, к более жестким требованиям к реализации монитора обращений для авторизации. Кроме того, важно отметить тот факт, что прикладные задачи и методы их решения с использованием рассматриваемых высокопроизводительных распределенных систем также могут накладывать ограничения на выбор средств обеспечения безопасности. Например, в силу важности ресурсоемких вычислительных задач, отдельного рассмотрения заслуживают вопросы влияния средств обеспечения безопасности на эффективность вычислений в Grid-среде.

По мнению авторов, следующие направления теоретических исследований и практических работ в области обеспечения безопасности в Grid-системах являются актуальными: разработка моделей нарушителя в применении к нескольким типовым процессам взаимодействия (информационным отношениям) в рамках решения задач с использованием Grid-систем; разработка программной архитектуры для обеспечения безопасности в типовых процессах решения задач с использованием Grid-систем; доработка существующих и разработка новых программных средств (механизмов, компонентов, сервисов, средств управления) и программных комплексов (комплексов средств, дистрибутивов операционных систем) для обеспечения безопасности в рамках предложенной архитектуры; проведение экспериментов, включая натурное моделирование действий нарушителя в рамках реализации процессов решения задач с использованием экспериментальных и виртуальных Grid-полигонов, а также демонстрацию преимуществ новых средств обеспечения безопасности.

Список литературы

1. **Humphrey M., Thompson M.** Security Implications of Typical Grid Computing Usage Scenarios // 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10 2001), 7—9 August 2001. San Francisco, CA, USA. IEEE Computer Society. 2001. P. 95—103.

2. **Sangroya A., Kumar S., Dhok J., Varma V.** Towards Analyzing Data Security Risks in Cloud Computing Environments // S. K. Prasad et al. (Eds.): ICISTM 2010, CCIS 54. Springer-Verlag Berlin Heidelberg. 2010. P. 255—265.

3. **Foster I., Kesselman C., Tuecke S.** The Anatomy of the Grid: Enabling Scalable Virtual Organizations // International J. Supercomputer Applications. 2001. 15 (3). URL: http://www.globus.org/alliance/publications/papers/anatomy.pdf

4. **Foster I., Kesselman C., Nick J., Tuecke S.** The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration // Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002. URL: http://www.globus.org/alliance/publications/papers/ogsa.pdf

5. **Foster I., Kessekan C., Tsudik G., Tuecke S.** A Security Architecture for Computational Grids // ACM Conference on Computer and Communications Security. 1998. P. 83—92.

6. **Галатенко В. А.** Основы информационной безопасности. Курс лекций / Под редакцией академика РАН В. Б. Бетелина. 4-е изд. М.: Интернет-Университет Информационных Технологий. БИНОМ. 2008. 205 с.

7. **Globus** Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective / The Globus Security Team, Edited by Von Welch. Version 4 updated September 12, 2005. URL: http://www.globus.org/toolkit/docs/4.0/security/GT4-GSI-Overview.pdf

8. **Globus** Toolkit 4.2.1: Security. The Globus Alliance. URL: http://www.globus.org/toolkit/docs/4.2/4.2.1/security/

9. **Globus** Toolit 5.0.2: Security. The Globus Alliance. URL: http://www.globus.org/toolkit/docs/5.0/5.0.2/security/

10. **Trusted** Computing Group. URL: http://www.trustedcomputinggroup.org/

11. **Loehr H., Ramasamy H. V., Sadeghi A.-R., Schulz S., Schunter M., Stueble C.** Enhancing Grid Security Using Trusted Virtualization // ATC LNCS. 2007. V. 4610. P. 372—384.

12. **Gentry C.** A fully homomorphic encryption scheme. Ph.D. thesis. Stanford University. 2009. URL: http://crypto.stanford.edu/craig

УДК 004.056.53

А. В. Шумилин, вед. науч. сотр.,
ОАО "НПО РусБИТех",
e-mail: shums@inbox.ru

Перспективный подход к обеспечению защиты информации от несанкционированного доступа в СУБД

Рассмотрен перспективный подход к обеспечению защиты информации от несанкционированного доступа в системах управления базами данных в составе автоматизированных систем в защищенном исполнении и приведен пример реализации этого подхода в системе управления базами данных отечественной операционной системы специального назначения Astra Linux Special Edition.

Ключевые слова: *системы управления базами данных, защита информации, автоматизированные системы, Astra Linux*

Введение

Согласно Концепции использования информационных технологий в деятельности федеральных органов государственной власти (ФОГВ) [1], актуальной задачей является разработка автоматизированных систем в защищенном исполнении (АС ЗИ) для использования в повседневной деятельности ФОГВ. Поскольку подобные программно-аппаратные комплексы предназначены для обработки информации, содержащей сведения, составляющие государственную тайну, к ним предъявляются требования по обеспечению защиты информации от несанкционированного доступа (НСД). Эти требования зависят от степени конфиденциальности информации и характера доступа к ней пользователей и приведены в руководящих документах (РД) ФСТЭК России [2, 3]. Выполнение требований должно подтверждаться сертификатом ФСТЭК России на ПО и АС.

Поскольку неотъемлемой частью системы обработки данных в АС являются системы управления базами данных (СУБД), они также должны отвечать требованиям по защите информации от НСД, применяемым к создаваемой системе в целом. Необходимо отметить, что специальных руководящих документов, предъявляющих требования к СУБД по обеспечению безопасности информации от НСД, в системе сертификации ФСТЭК не существует. Таким образом, СУБД должна в первую очередь соответствовать требованиям РД для средств вычислительной техники (СВТ) [2]. Однако СУБД и клиентское ПО функционируют под управлением ОС. Сертификация СУБД отдельно по требованиям СВТ может в результате привести к невозможности выполнения требований РД для АС. Следовательно, реализацию требований по обеспечению безопасности информации от НСД целесообразно осуществлять на основе интеграции с механизмами защиты информации ОС. Также это позволяет переложить на ОС часть требований по защите информации от НСД, поскольку не все из них могут быть напрямую применены к СУБД.

К одним из необходимых требований к АС, обрабатывающим информацию, содержащую сведения, составляющие государственную тайну, относится обеспечение мандатного разграничения доступа (МРД) к информации во всем цикле ее обработки. И если дискреционное разграничение доступа (ДРД) к информации в СУБД существует, то механизмы мандатного разграничения доступа являются нестандартным решением, требующим отдельной разработки.

Рассмотрим, как может быть обеспечено выполнение в СУБД требований по защите информации от НСД, какие при этом возникают сложности и как они могут быть преодолены.

Существующие решения

Существуют СУБД с закрытыми исходными кодами, обеспечивающие защиту от НСД и в том числе с использованием МРД, но реализация этих механизмов также закрыта. Например, многие известные СУБД, такие как Informix, Microsoft SQL Server, Oracle, Sybase сертифицированы для создания защищенных систем обработки данных по требованиям Агентства Национальной Безопасности США [4]. Как правило, подобные варианты исполнения получают приставку *Trusted*. Указанные продукты поставляются без исходного кода, созданы по зарубежным требованиям и не могут быть сертифицированы в России для работы с информацией, содержащей сведения, составляющие государственную тайну [5].

В РФ существует концепция развития разработки и использования свободного ПО, в соответствии с которой разработан и утвержден правительством План перехода федеральных органов исполнительной власти и федеральных бюджетных учреждений на использование свободного программного обеспечения [6]. В связи с этим интерес могут представлять средства обеспечения защиты информации в информационных системах с открытыми исходными текстами. Одним из самых известных средств является расширение SELinux [7], добавляющее системам на основе Linux средства разграничения доступа, удовлетворяющие разнообразным требованиям, в том числе и к МРД. Решение является универсальным, позволяющим определять политику разграничения доступа, подходящую для конкретной создаваемой информационной системы. Концепция заключается в использовании правил, описывающих отношения между такими понятиями, как сущности, роли, домены и диапазоны меток. Аналогичное расширение есть и для СУБД PostgreSQL [8]. Основной недостаток этих систем заключается в сложности создания политики безопасности и дальнейшего администрирования. Другие расширения типа AppArmor [9, 10], Tomoyo Linux [11] и RSBAC [12], на наш взгляд, рассматривать нецелесообразно, поскольку они проигрывают по своим возможностям SELinux и не в полной мере соответствуют требованиям РД.

В РФ сертифицированными СУБД по требованиям РД являются продукты серии "Линтер" (разработчик — ЗАО "НПП РЕЛЭКС") [13, 14]. Эти СУБД являются примером защищенных серийно поставляемых изделий, реализация требований безопасности информации в которых предусматривает полную независимость от механизмов защиты информации ОС или каких-либо других средств защиты информации, применяемых в составе информационных систем. Тем самым, создание защищенной и интегрированной с ОС СУБД является, безусловно, актуальной задачей, решение которой должно учитывать как особенности функционирования СУБД, так и особенности реализации требований безопасности информации в ОС.

Особенности организации разграничения доступа в СУБД

Основное назначение СУБД состоит в предоставлении пользователям АС одновременного доступа к данным наиболее эффективным и быстрым способом. Это достигается за счет механизма транзакций, который обеспечивает прозрачный одновременный доступ к данным, сохраняя при этом данные в согласованном состоянии. Помимо этого существует целый ряд механизмов обеспечения надежности работы СУБД и поддержки эффективности доступа к данным, которые реализованы на уровне ядра СУБД. Таким образом, для обеспечения минимальных потерь в надежности и производительности необходима реализация средств разграничения доступа к данным и обеспечения информационной безопасности непосредственно в ядре СУБД.

Отдельно следует отметить следующие особенности реализации механизмов защиты информации от НСД в СУБД:

1. Обработка данных одного пользователя осуществляется в рамках сеанса работы с СУБД с использованием внутренних субъектов доступа, в связи с чем необходимо сопоставление пользователя ОС с сеансом обработки данных и внутренними субъектами доступа.

2. Ресурсы, защищаемые в СУБД с использованием МРД, не являются объектами файловой системы, доступ к которым может быть разграничен с использованием механизмов ОС. В СУБД должен быть свой механизм, интегрированный с механизмами ОС для обеспечения МРД во всем цикле обработки информации (ввод метки на клиенте, распространение метки при доступе по сети средствами ОС и непосредственно разграничение в СУБД).

3. СУБД обладают механизмами ДРД. В целях обеспечения требования одновременного санкционирования доступа механизмами ДРД и МРД, реализация МРД должна быть исполнена в тесной связи с существующим механизмом ДРД. При этом необходимо учитывать наличие большого числа объектов доступа и разных видов доступа к ним, которые при реализации МРД необходимо транслировать в операции чтения и записи.

4. В СУБД содержатся системные механизмы по обслуживанию БД, включая резервное копирование и восстановление, которые должны функционировать, не нарушая требований к защите и разграничению доступа.

5. Реализация всех механизмов защиты информации от НСД, включая МРД, должна минимально влиять на производительность и надежность работы.

6. Помимо существующих средств журналирования и отладки должна быть обеспечена регистрация событий доступа согласно требованиям РД.

Остановимся подробнее на этих особенностях.

Аутентификация и сопоставление пользователей

Для работы механизмов разграничения доступа требуется надежное сопоставление пользователя с устройством, что в отношении СУБД может трактоваться как требование сопоставления пользователя АС с сеансом обработки данных. Это достигается использованием различных механизмов надежной идентификации и аутентификации и изоляцией сеансов работы с данными. В настоящее время практически все СУБД поддерживают сетевой протокол аутентификации Kerberos, который является общепринятым стандартом организации безопасной аутентификации в незащищенных сетях.

В рамках сеанса обработки данных могут использоваться внутренние субъекты доступа СУБД, такие как группы, роли или хранимые процедуры. Указанные субъекты используются в основном для облегчения управления дискреционными правилами разграничения доступа и не должны искажать мандатного контекста самого пользователя. Таким образом, мандатный контекст пользователя должен быть непосредственно связан именно с сеансом обработки данных и не зависеть от сопоставляемых с ним внутренних субъектов доступа. Также очевидно, что мандатный контекст должен быть получен при инициализации сеанса связи и проверен средствами ОС.

Дискреционное разграничение доступа

В системах управления базами данных ДРД к объектам баз данных (БД) было разработано в 1992 г. и нашло свое отражение в спецификации языка доступа к данным SQL ревизии SQL-92. К настоящему времени дискреционный доступ реализован практически во всех СУБД, отличия в основном связаны с числом типов защищаемых объектов и разнообразием детализации способов доступа к ним.

В случае БД субъектом является пользователь или пользовательский процесс, а объектом — одна из сущностей внутреннего представления данных или вычислительных ресурсов в СУБД (например, таблица, представление, хранимая процедура и т.п.). Это связано с тем, что в общем случае отдельная строка таблицы не является однозначно идентифицируемым объектом и соответственно дискреционные правила разграничения доступа к ней применены быть не могут.

В некоторых СУБД к субъектам могут относиться процедуры обработки данных, роли и группы пользователей. К типам доступа относятся существующие базовые операции языка запросов SQL, такие как: CONNECT, SELECT, INSERT, UPDATE, DELETE, DROP, CREATE; а также, в зависимости от конкретной СУБД, операции ссылочной целостности и иные специфичные для рассматриваемой СУБД операции. Для предоставления прав и их отбора используются команды SQL GRANT и REVOKE.

Также язык запросов и манипулирования данными SQL позволяет реализовывать несколько способов расширения ДРД. Первым из них является объединение пользователей в группы или роли, что позволяет значительно снизить затраты на администрирование. Второй способ связан с организацией доступа к данным не непосредственно через доступ к таблицам БД, а с помощью механизма представлений и хранимых процедур. В этом случае пользователь получает доступ только к определенным образом отобранным данным. Это способ очень широко используется при проектировании и разработке БД.

Также разграничение доступа может осуществляться путем модификации запроса пользователя во время исполнения. При этом может меняться набор данных, к которым предоставляется доступ. Еще одно название такого способа — "Детальный контроль доступа", один из вариантов которого предлагается в СУБД Oracle [13].

В настоящий момент реализация ДРД в большинстве СУБД полностью удовлетворяет предъявляемым РД требованиям.

Мандатное разграничение доступа

Поскольку МРД применяется в основном для построения систем специального назначения, общего решения на уровне стандартов и спецификаций языка запросов не существует. Каждый из производителей СУБД при необходимости встраивает в свою систему собственное решение. Например, Oracle Label Security [15, 16].

Особенностью встраивания механизма МРД в СУБД является большой набор типов доступа субъектов к объектам, в то время как МРД оперирует только понятиями чтения и записи информации. И основная задача состоит в привидении используемых типов доступа к этим операциям, что может быть сделано следующим образом:

- INSERT — доступ на запись;
- UPDATE, DELETE — последовательное выполнение доступа на чтение и запись (обусловлено тем, что перед выполнением операции необходимо найти информацию, над которой проводится операция);
- SELECT — доступ на чтение.

Аналогичным образом трактуются и операции модификации схемы данных, т.е. создание и манипулирование объектами БД:

- CREATE, ADD — доступ на запись;
- ALTER, DROP — последовательное выполнение доступа на чтение и запись.

Как видно, для выполнения операций модификации или удаления необходимо точное соответствие мандатного контекста пользователя метке конфиденциальности объекта.

В реляционной модели в качестве структуры, обладающей меткой, естественно выбрать кортеж, по-

скольку именно на этом уровне детализации осуществляются операции чтения-записи информации в СУБД [16]. При этом местом хранения метки может быть выбран только сам кортеж, только таким образом метка будет неразрывно связана с данными, содержащимися в нем. Также это позволяет обеспечить корректность обработки информации при выполнении служебных операций в СУБД, таких как индексация, оптимизация, резервное копирования и т.п.

В СУБД МРД реализуется, как правило, расширением механизма разграничения доступа дополнительными возможностями, введением дополнительного типа данных, соответствующего метке конфиденциальности, добавлением необходимых атрибутов объектам БД в системном каталоге и расширением языка запросов SQL необходимыми командами декларативного управления правилами разграничения доступа (ПРД).

Целостность и надежное восстановление

Важным требованием является обеспечение целостности средств защиты информации в составе СУБД, а также поддержка функций надежного восстановления данных.

Эти задачи связаны с наличием в СУБД средств резервного копирования и восстановления данных, а также механизмов восстановления после сбоев. В большинстве систем это реализуется выгрузкой данных из БД в архивную копию и ведением журнала транзакций. В случае использования МРД необходимо обеспечить корректное сохранение и восстановление мандатных атрибутов данных и объектов БД, а журнал транзакций должен фиксировать все действия с мандатными атрибутами данных и действиями по изменению правил мандатного разграничения доступа.

Целостность на уровне исполняемых модулей может быть обеспечена механизмами контроля целостности ОС, а целостность атрибутов данных — механизмами контроля целостности БД, реализованными в СУБД. При этом особенно важна корректность реализации механизмов индексирования и оптимизации БД, которые должны гарантировать неизменность мандатных атрибутов данных при выполнении любых операций обслуживания БД.

Регистрация событий

В соответствии с РД должна осуществляется регистрация событий идентификации и аутентификации, предоставления доступа к объекту, его создания и уничтожения, а также действий по изменению правил разграничения доступа. При этом для каждого события указывается подробная информация о времени наступления, типе события, задействованных субъектах и объектах и результате предоставления доступа (успешность).

Штатные средства ведения журналов работы СУБД не в полной мере отвечают этому требованию, так как предназначены больше для отладки и контроля функционирования и могут не обеспечивать требуемого уровня детализации и надежности фиксации событий. В связи с этим, как правило, необходима модификация СУБД. Это связано и с тем, что защищенная СУБД должна функционировать в среде защищенной ОС, и в этом случае регистрация событий СУБД должна вестись с помощью отвечающих за это механизмов ОС.

Защищенная СУБД ОС СН Astra Linux Special Edition

Рассмотрим способы решения перечисленных вопросов на примере защищенной СУБД, входящей в состав операционной системы специального назначения Astra Linux Special Edition [17], в разработке средств защиты информации которой автор принимал непосредственное участие. За основу при выполнении работ взята СУБД PostgreSQL.

Основанием выбора PostgreSQL стало то, что она относится к классу промышленных СУБД и способна решать задачи обработки данных любых масштабов. При этом доступны исходные тексты СУБД, а ее разработчиками осуществляется ее непрерывное сопровождение, развитие, устранение ошибок и улучшение функциональных возможностей. К тому же она обладает прозрачной и понятной архитектурой, позволяющей качественно проводить необходимые доработки. Остановимся на них подробнее.

В соответствии с РД [2] должен существовать диспетчер доступа, осуществляющий перехват всех обращений субъектов к объектам и разграничение доступа в соответствии с заданным принципом разграничения доступа. При этом не должно существовать возможности доступа к защищаемым данным в обход диспетчера доступа. Такой диспетчер доступа реализован в ядре СУБД. При этом защищенная СУБД интегрирована со встроенными средствами защиты ОС Astra Linux Special Edition не только в целях решения задач идентификации—аутентификации, но и в целях использования мандатных атрибутов пользователей, задаваемых в ОС, механизмов регистрации событий и разграничения доступа к объектам файловой системы, содержащих физическое представление информации, хранящейся в СУБД.

Обеспечение сопоставления пользователя сеансу

В случае использования защищенной СУБД в составе защищенной ОС, множество пользователей, имеющих право работы с СУБД, является подмножеством пользователей ОС. Поэтому помимо прав доступа, определенных и хранящихся непосредственно

в СУБД, для пользователя, после прохождения им процедуры аутентификации, для проверки и разграничения доступа используются мандатные метки и специальные мандатные атрибуты, присвоенные ему в ОС.

СУБД обеспечивает параллельное выполнение запросов пользователей в различных сеансах связи с БД. При этом для каждого сеанса работы с пользователем СУБД создается отдельный процесс ОС, в пределах которого осуществляется работа с информацией пользователя. В случаях потери соединения клиента с сервером, соединение принудительно завершается, и процесс осуществляет регламентированные действия в этой ситуации. После этого не существует возможности повторной установки соединения с этим процессом, что гарантирует невозможность подключения к СУБД без похождения процедур идентификации и аутентификации. Поскольку каждый процесс запускается в отдельном виртуальном пространстве, управляемом диспетчером памяти ОС, гарантии изоляции модулей обеспечивает ОС. Таким образом, не существует возможности доступа одного пользователя к данным другого пользователя при одновременной работе с СУБД.

Обеспечение разграничения доступа

В качестве механизма дискреционного разграничения доступа используется штатный механизм PostgreSQL с возможностью дополнительного управления с помощью параметров разрешением делегирования прав доступа владельцем и пакетного вывода данных из БД. При этом доступ предоставляется только при одновременном соответствии установленным правилам МРД.

Также, в PostgreSQL, начиная с версии 8.4, объектами ДРД могут являться столбцы объектов, поскольку могут быть однозначно идентифицированы по составному имени объекта и столбца, так как имя столбца внутри объекта является уникальным. Во избежание неоднозначности трактовки правил разграничения доступа, разграничение доступа по столбцам проводится только в случае отсутствия явно заданных правил разграничения доступа к объекту в целом.

Проверка мандатных прав доступа к таблицам и видам осуществляется одновременно с проверкой дискреционных прав доступа к ним, после разбора и построения плана запроса, непосредственно перед его выполнением, когда определены все необходимые для проверки данные и проверяемые объекты. Проверка мандатных прав доступа к записям таблиц осуществляется в процессе выполнения запроса при последовательном или индексном сканировании данных. Указанные точки контроля проходятся при выполнении любых запросов, и, следовательно, не могут быть обойдены. Защищенная СУБД не имеет собственного механизма назначения, хранения и модификации меток пользователей и использует для этого механизмы ОС.

Как было сказано ранее, существует возможность разграничения доступа по столбцам объектов. В этом случае во избежание неоднозначности мандатные правила разграничения доступа применяются аналогично дискреционным либо на уровне объекта, либо на уровне столбцов. Вариант применения правил МРД может задаваться для каждого объекта в отдельности.

Для реализации МРД в СУБД был добавлен тип данных, отвечающий метке конфиденциальности, и определены операции сравнения и индексирования информации этого типа. Список атрибутов объектов БД в системном каталоге расширен полем этого типа, а также дополнительными информационными атрибутами, отвечающими за управление правилами разграничения доступа. При создании объекта БД может быть указано использование меток конфиденциальности строк объекта. В этом случае физическое представление объекта на диске будет содержать метку конфиденциальности в каждой строке данных. Декларативное управление правилами разграничения доступа осуществляется с помощью языка запросов SQL, синтаксис которого был изменен для возможности управления мандатными правилами разграничения доступа.

При установке соединения определяется мандатный контекст пользователя, который назначается текущему сеансу работы с БД, и в дальнейшем все операции с данными осуществляются в этом мандатном контексте. Таким образом, на уровне работы с записями, данные будут получать метку конфиденциальности, соответствующую мандатному контексту пользователя.

Для администратора БД предусмотрены системные привилегии игнорирования мандатного разграничения доступа, только таким образом могут быть проведены регламентные работы с БД (например, восстановление резервной копии), так как это требует возможности установки меток данных, сохраненных ранее.

Обеспечение целостности и надежного восстановления

Диспетчер доступа реализован как часть ядра СУБД и получает управление на ранних этапах запуска сервера. Только после этого запускаются процедуры обслуживания запросов пользователей.

Целостность СУБД эквивалентна целостности программного кода, реализующего диспетчер доступа, и целостности атрибутов безопасности защищаемых объектов БД. Целостность программного кода обеспечивается средствами контроля целостности по контрольным суммам, реализованными в защищенной ОС. Целостность атрибутов безопасности прове-

ряется при запуске сервера как часть общей комплексной проверки целостности данных.

Для обеспечения возможности надежного восстановления после сбоев в СУБД предусмотрен механизм журналирования всех изменений данных в процессе работы. Так как атрибуты безопасности также являются данными, хранящимися в БД, их изменения также сохраняются в журнале транзакций.

Регистрация и учет событий

В соответствии с РД регистрируются все попытки доступа к защищаемой информации (к именованным объектам доступа). Подсистема аудита является неотъемлемой частью ядра сервера и регистрирует все регламентированные события с помощью средств регистрации событий защищенной ОС. При этом существует возможность настройки состава регистрируемых событий.

Заключение

Исследование и анализ особенностей реализации средств защиты информации от НСД в СУБД в сочетании с использованием мирового опыта разработки свободного ПО позволили решить задачу по созданию СУБД, которая в составе ОС специального назначения соответствует требованиям РД. Таким образом, с учетом утвержденных на государственном уровне Концепции использования информационных технологий [1] и Плана перехода федеральных органов исполнительной власти и федеральных бюджетных учреждений на использование свободного программного обеспечения [6], можно утверждать, что применение такой защищенной СУБД в составе ОС Astra Linux Special Edition позволяет создавать информационные системы, обрабатывающие информацию, содержащую сведения, составляющие государственную тайну.

Защищенная СУБД в составе ОС Astra Linux Special Edition получила в системе сертификации Минобороны России сертификат на соответствие требований РД по классу 3 защищенности СВТ и уровню 2 контроля отсутствия недекларированных возможностей, в настоящее время ведется работа по сертификации в ФСТЭК.

Несмотря на успешное решение этой задачи, остается еще много вопросов, связанных, например, с обеспечением необходимых механизмов защиты информации от НСД при создании программно-аппаратных комплексов для построения отказоустойчивых кластеров баз данных и решений по увеличению производительности за счет параллельного выполнения запросов и балансировки нагрузки.

Список литературы

1. **Концепция** использования информационных технологий в деятельности федеральных органов государственной власти до 2010 года. "Российская газета" — Федеральный выпуск № 3597, 7 октября 2004.

2. **Гостехкомиссия России.** Руководящий документ. Средства вычислительной техники. Защита от несанкционированного доступа к информации. Показатели защищенности от несанкционированного доступа к информации. М.: Военное издательство, 1992.

3. **Гостехкомиссия России.** Руководящий документ. Автоматизированные системы. Защита от несанкционированного доступа к информации федеральных органов исполнительной власти и федеральных бюджетных учреждений на использование свободного программного обеспечения. Классификация автоматизированных систем и требования по защите информации. М.: Военное издательство, 1992.

4. **Trusted** Database Management System Interpretation. National Computer Security Center. NCSC-TG-021 Version 1, April 1991.

5. **Государственный** реестр сертифицированных средств защиты информации Системы сертификации средств защиты информации по требованиям безопасности информации № РОСС RU.0001.01БИ00. URL: http://www.fstec.ru/_doc/reestr_sszi/ _reestr _sszi.xls

6. **План** перехода федеральных органов исполнительной власти и федеральных бюджетных учреждений на использование свободного программного обеспечения на 2011—2015 годы. URL: http://tp-npp.ru/index.php?option=com_content&view=article&id=102:-2011-2015-&catid=31:acts&Itemid=18

7. **Security-Enhanced** Linux. National Security Agency, 2009. URL: http://www.nsa.gov/research/selinux/

8. **SEPostgreSQL** Specifications. URL: http://wiki.postgresql.org/wiki /SEPostgreSQL_Specifications

9. **AppArmor** vs. SELinux // Linux Magazin 2006. ISSUE 69: August. URL: http://www.linux-magazine. com/w3/issue/69/AppArmor_ vs_SELinux.pdf

10. **Novell** AppArmor, Official documentation. URL: http://www.novell.com/documentation/apparmor/

11. **TOMOYO** Linux 2.3.x: The Official Guide. URL: http://tomoyo.sourceforge.jp/2.3/index.html.en

12. **Rule** Set Based Access Control, RSBAC Documentation. URL: http://www.rsbac.org/documentation

13. **СУБД** Линтер, Официальный сайт. URL: http://www.lintersql.com/ru/about/specification/

14. **Маркин С. П., Бойченко И. А., Максимов В. Е., Козленко Л. А.** Защищенная реляционная СУБД Линтер // Открытые Системы. 1999. № 11—12. URL: http://www.osp.ru/os/1999/11-12/177904

15. **Feuerstein S.** Oracle PL/SQL Programming Guide to Oracle 8i Features. O'Reilly Media, Inc. 1999.

16. **Смирнов С. Н.** Безопасность систем баз данных. М.: Гелиос АРВ, 2007.

17. **Astra** Linux Special Edition, НПО РусБиТех. URL: http://astra-linux.com/

УДК 004:378

В. В. Костюк, канд. техн. наук, проф., **Д. А. Пантелеева,** студент,
Российский государственный университет
инновационных технологий и предпринимательства,
e-mail: Viacheslav.Kostyuk@itbu.ru

Использование платформы IBM Jazz в дипломном проектировании

Рассматриваются вопросы управления дипломным проектированием студентов, разработки и документирования дипломных работ выпускников учебных заведений. Использование системы Jazz обеспечивает коллективную работу над проектами, повышая эффективность выполнения проектов. В данном случае коллектив состоит из нескольких участников, работающих над дипломным проектом: руководителями, консультантами разных частей диплома и студентом, реализующим этот диплом.

Ключевые слова: *документирование разработки, дипломные проекты, коллективы, артефакты, планирование заданий, распределенная разработка, программная инженерия*

Средства платформы IBM Jazz от компании IBM Rational предназначены для организации и управления коллективной разработкой программных продуктов и документирования их [1, 2]. IBM Jazz предоставляет распределенной географически группе разработчиков удобные средства для организации эффективного процесса коллективной работы [3, 4].

Платформа Jazz, выдвинутая компанией IBM Rational на рынок централизованных систем управления изменениями и версиями, обеспечивает возможность организации коллективной работы над проектами разного характера, и в первую очередь ориентирована на разработку программных продуктов, охватывая все процессы, связанные с разработкой программного обеспечения, такие как:

- проектирование и разработка документации;
- выпуск документации;
- сопровождение документации.

Платформа Jazz предназначена для использования не только в организациях, специализирующихся на разработках программного обеспечения. Она плодотворно может быть использована и для студенческих коллективов, обучающихся по специальности "Информационные системы и технологии", изучающих дисциплины, связанные с программированием [5].

В данной статье рассматриваются вопросы управления разработкой и документированием дипломных работ выпускников учебных заведений. Использование системы Jazz вполне обеспечивает работу коллектива из нескольких участников, работающих над дипломным проектом: руководителями разных частей диплома и студентом, реализующим этот диплом. Руководители дипломного проекта готовят индивидуальное задание на выполнение этого проекта. Эти задания представляются в проекте системы Jazz как задания типа "Задача". Проект системы Jazz — это поименованная рабочая область системы Jazz, которая выделяется в нашем случае для работы с дипломным проектом. На основе этих заданий и созданного графика работ формируется план выполнения дипломной работы, который просматривается студентом, выполняющим дипломную работу. План, в свою очередь, отслеживается руководителями.

Материалы выполненной пояснительной записки размещаются в артефактах данного проекта Jazz, корректируются, совершенствуются, дорабатываются студентом к защите. После защиты лучшие диплом-

ные проекты остаются в системе Jazz для свободного доступа другим студентам в целях использования опыта и знаний в своих исследовательских и практических работах.

Планируется, что в Российском государственном университете инновационных технологий и предпринимательства студенты уже второго года обучения смогут в среде Jazz знакомиться с материалами дипломных работ в целях осознания того, каких результатов они должны будут достичь после обучения в нашем вузе. В системе Jazz будет формироваться фонд выпускных квалификационных работ по специализации кафедры "Информационные системы": проектирование информационных систем, администрирование информационных систем, сетевые технологии и другие направления.

Для документирования проекта используются артефакты таких составляющих системы Jazz, как "область проекта", "область коллектива", "задание".

Среда платформы Jazz представлена на рис. 1 (см. третью сторону обложки).

Дипломная работа является строго структурированным документом, представляющим собой пояснительную записку к диплому, которая создается студентом в соответствии с требованиями, изложенными в методическом указании "Дипломное проектирование".

Каждый дипломный проект идентифицируется шифром проекта (не более восьми символов), который определяет руководитель проекта при создании области коллектива. Он же формирует в области проекта график реализации проекта, категории функциональности заданий коллектива, определяет задачи проекта, формирует задания и на основе их создает планы реализации проекта.

Инструктивный материал по работе в системе Jazz сконцентрирован в особом проекте JAZZ, доступ к которому осуществляется по запросу общего пользования "Путеводитель пользователя Jazz".

Функциональности заданий коллектива определяются следующими категориями:

- пояснительная записка;
- учебная практика;
- практика реализации;
- задания на разработку.

Эти же категории используются для названий запросов на доступ к заданиям этих категорий. Задания, сгруппированные в последнем из этих запросов, являются заданиями типа "Задача", которые ставятся перед дипломником. Перечень этих заданий определен документом "Индивидуальное задание к дипломному проекту". Первые три запроса обеспечивают доступ к документируемому материалу. Например, запрос "Учебная практика" обеспечивают доступ к материалам, которыми пользуется студент при выполнении дипломной работы. Это разного рода методички, лабораторные работы. Здесь же можно найти и методичку "Дипломное проектирование".

Рассмотрим артефакты системы Jazz, которые используются для размещения разделов дипломного проекта.

Область проекта в системе Jazz (рис. 2).

Из всех артефактов области проекта для документирования используются:

- Название проекта — общий шифр проекта, в рамках которого выполняются однотипные дипломные работы.
- "Сводка" — название дипломного проекта.
- "Описание" — более подробные сведения о назначении данного проекта. В этом артефакте описывается, какого типа дипломные работы выполняются в рамках данного проекта, например с использованием Java технологий или проекты по направлению администрирования информационных систем, а также описывается, какими средствами, какими участниками (исполнителями и руководителями дипломных работ) в каждом коллективе выполняется данный диплом.

Область коллектива в системе Jazz (рис. 3, см. четвертую сторону обложки).

Из всех рассмотренных артефактов области коллектива для документирования используются:

- Название коллектива — шифр темы дипломного проекта.
- "Сводка" — краткое название задачи, возложенной на коллектив. В этом артефакте размещается название темы дипломного проекта.
- "Описание" — более подробные сведения о решаемой коллективом задаче. В этом артефакте размещается аннотация к дипломному проекту.

Задание типа "Задача" в системе Jazz представлено на рис. 4.

Задание типа "Прецедент" (задание-прецедент) в системе Jazz фрагментарно представлено на рис. 5, см. четвертую сторону обложки.

Отличие заданий типа "Задача" от заданий типа "Прецедент" заключается в том, что первые предназначены для размещения в нем названий пунктов из индивидуального задания на дипломное проектирование. Как правило, это глагольные выражения, требующие выполнения работ. Тогда как вторые предназначены для документирования выполняемых работ.

Задания типа "Прецедент" с позиций дипломного проекта рассматриваются в основном как прецеденты-сценарии. Прецеденты-сценарии отводятся для крупных разделов дипломного проекта и всего диплома в целом. Подразделы небольших разделов специальной части дипломного проекта размещаются в атомарных прецедентах. Атомарные прецеденты фигурируют только в ссылках к прецедентам-сценариям как дочерние прецеденты.

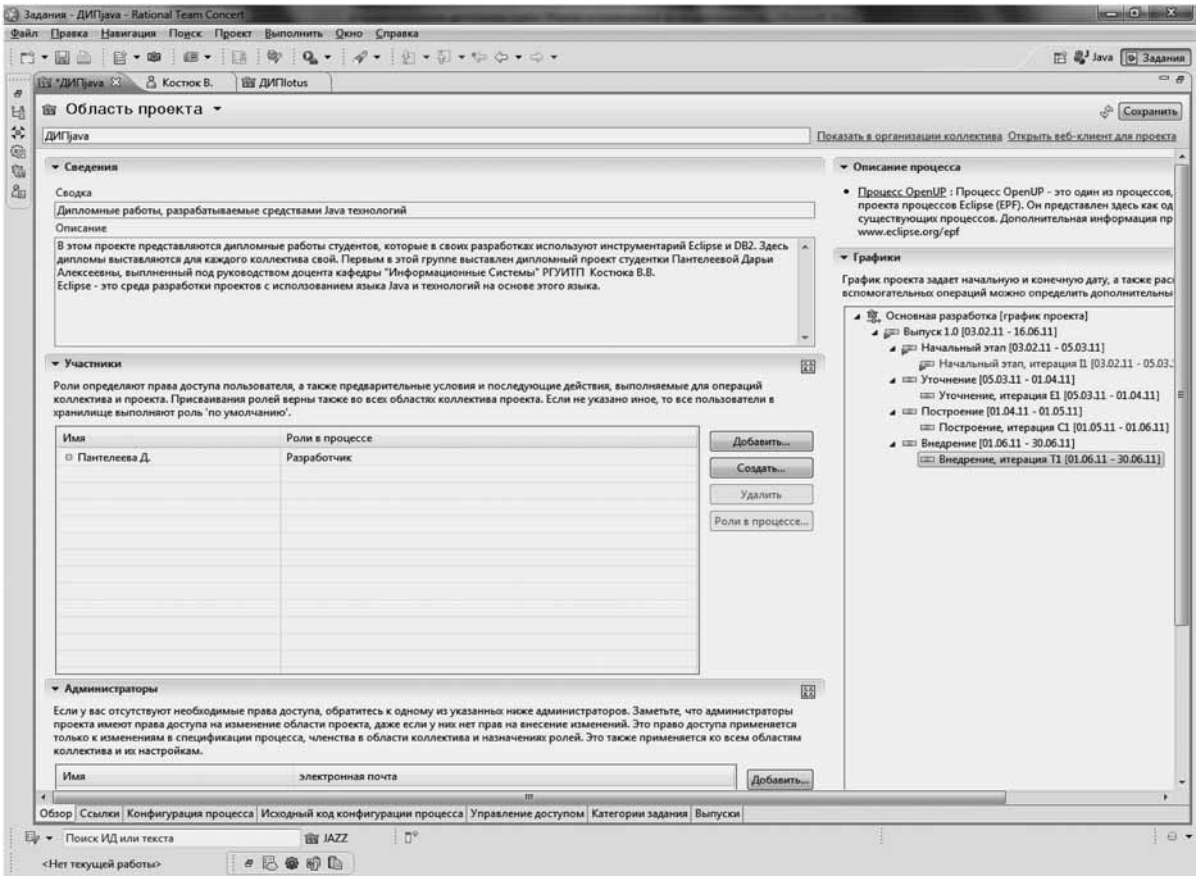

Рис. 2. Область проекта в системе Jazz

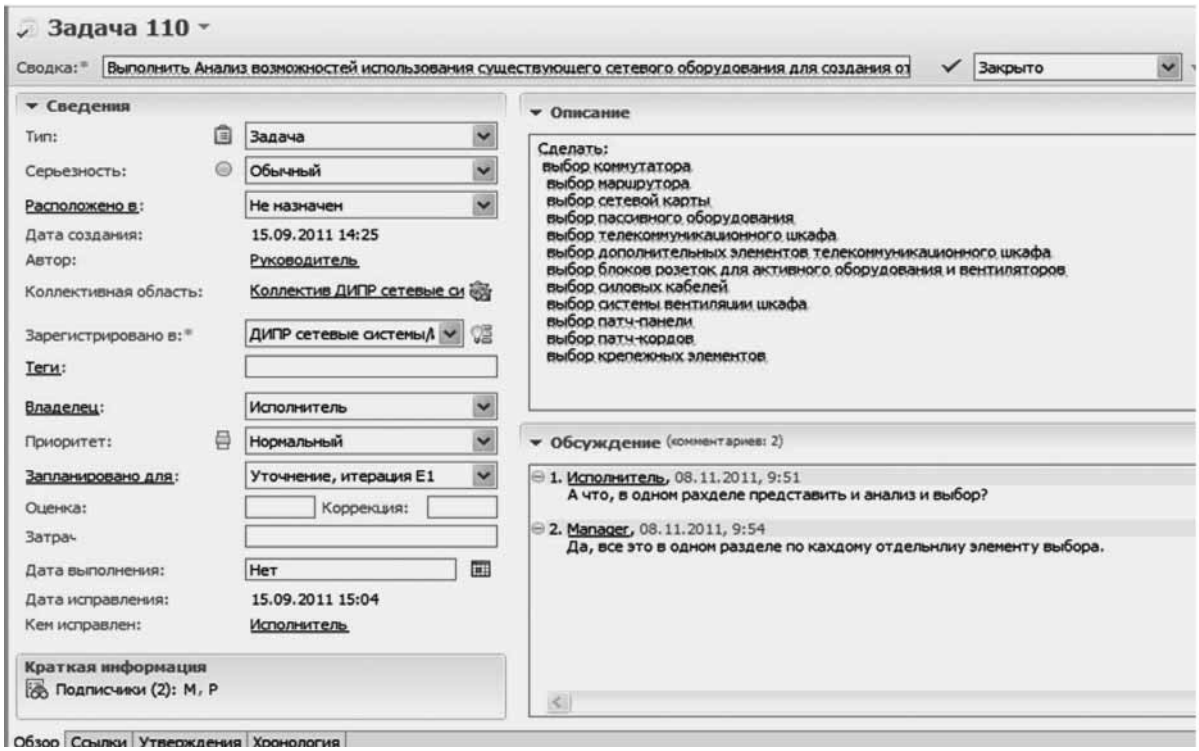

Рис. 4. Задание типа "Задача"

Для документирования разделов (подразделов) дипломного проекта артефакты задания-прецедента используются следующим образом.

- "Сводка" — краткая идентификация прецедента, название раздела (подраздела) специальной части дипломного проекта. Для диплома в целом это тема дипломного проекта.
- "Сведения" — вся необходимая информация, характеризующая данный прецедент.
- "Описание" — подробные сведения о назначении данного прецедента. В этом артефакте размещаются пункты данного раздела (подраздела) из содержания диплома; для диплома в целом это аннотация к дипломному проекту. В заданиях типа "Задача" этот артефакт предназначен для описания условий выполнения дипломного проекта.
- Вложения, которые можно добавить, удалить, сохранить, переименовать. Вложения — это изображения, схемы, текстовые файлы, которые могут быть прикреплены к прецеденту в проекте. В качестве прикрепляемого файла выступает файл с текстом раздела (подраздела) специальной части дипломного проекта. Для дипломного проекта в целом в качестве прикрепляемого файла выступает весь дипломный проект целиком с титульным листом.

Пример прикрепления файла вложения к заданию и создания ссылок к ассоциативным заданиям приведен на рис. 6. На рис. 7 приведен пример прикрепленного к заданию-прецеденту "Моделирование предметной области" файла, в котором представлена модель жизненного цикла изделия.

- Ссылки, которые можно добавить, открыть или удалить из проекта. Ссылки представляют собой другие задания, связанные с данным прецедентом. С помощью ссылок можно выстроить иерархические или другого типа связи с другими заданиями-прецедентами. В данном случае создаются связи с дочерними прецедентами-сценариями, роль которых выполняют подразделы специальной части дипломного проекта или с сопутствующими (связанными) файлами-прецедентами. Для дипломного проекта в целом это ссылки на задания-прецеденты, связанные с вопросами актуальности, целей, проблем из дипломного проекта.
- Утверждение. Под утверждением понимается решение руководителя дипломного проекта о принятии работ по данному прецеденту. Для разделов (подразделов) — это выводы (если они есть в дипломе).

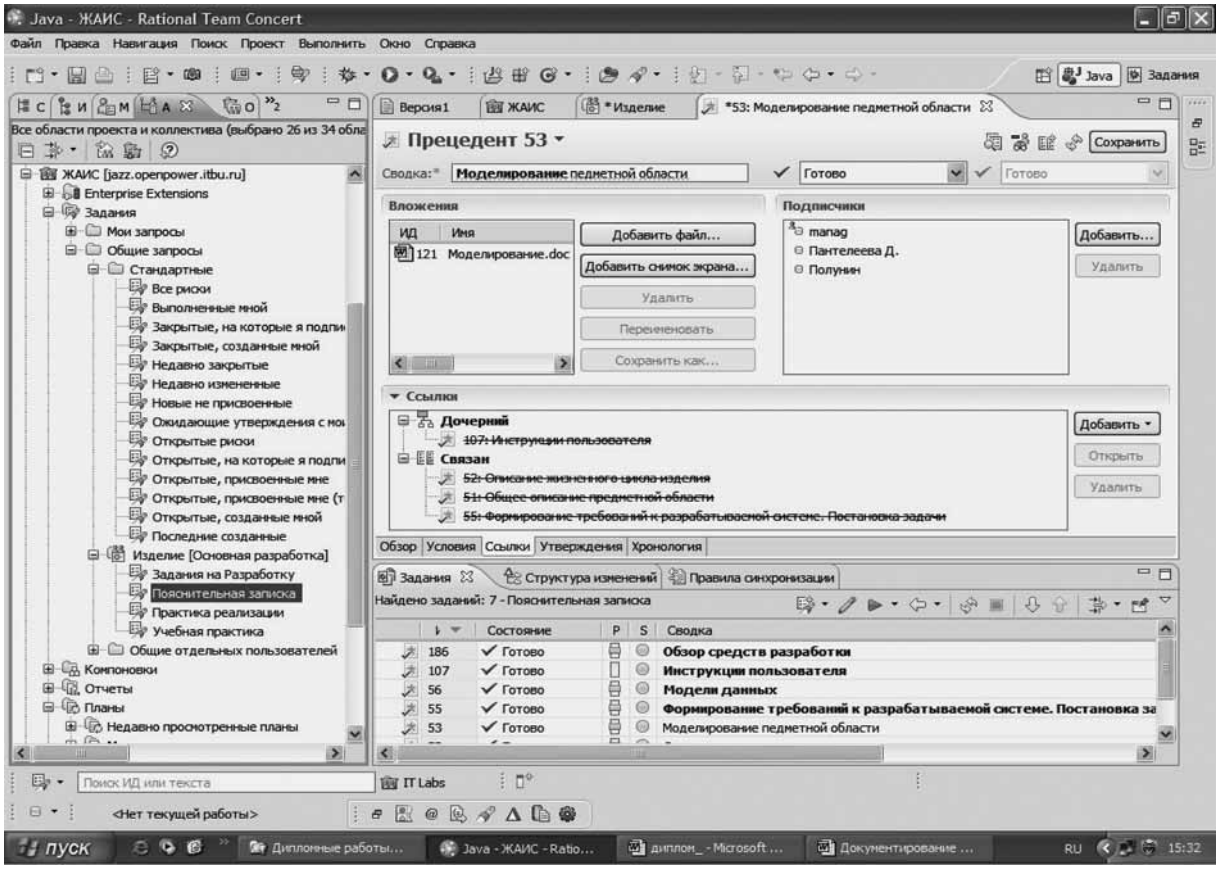

Рис. 6. Артефакты "Вложения" и "Ссылки" в Задании системы Jazz

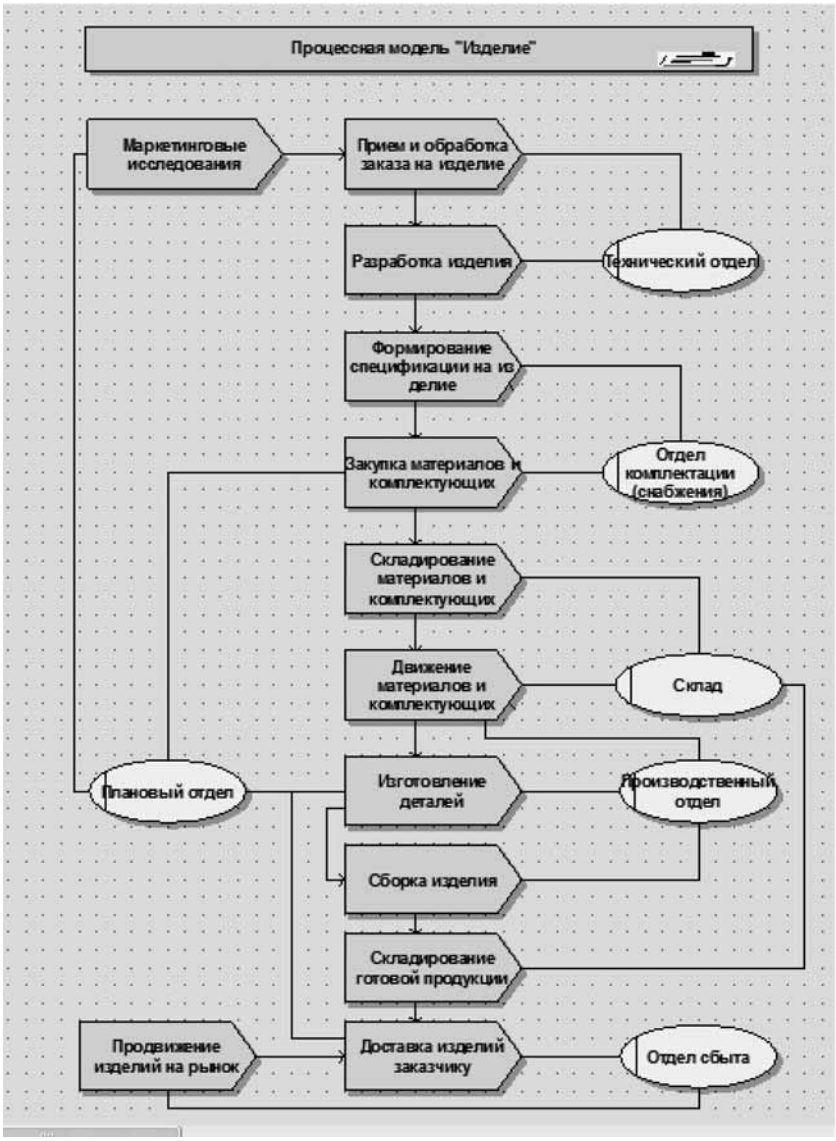

Рис. 7. Фрагмент файла, прикрепленного к заданию-прецеденту, представляющего бизнес-модель "Жизненный цикл изделия"

Для дипломного проекта в целом — это раздел "Заключение" дипломного проекта.

Есть одно стандартное задание, которое автоматически формируется при создании области проекта — это задание-задача "Определение видения". Его также можно использовать для документирования. Для дипломных работ в описании этого задания можно показать постановку задач из дипломного проекта.

Аналогичным образом документируются такие разделы, как технологическая, экономическая части проекта и приложения.

Раздел "Литература" формируется отдельным заданием-прецедентом с таким же названием в артефакте "Сводка".

Все задания-прецеденты объединяются единым запросом "Пояснительная записка".

Запросом "Учебная практика" объединяются задания-прецеденты, связанные с учебно-методическими материалами, лабораторными работами, на основе которых выполнялся дипломный проект.

Запросом "Практика реализации" объединяются задания-прецеденты, связанные с содержанием технологической части дипломной работы, где в основном демонстрируется, как выполнялся дипломный проект, какими методами, средствами, приемами, как получались результаты.

Запросом "Задания на разработку" объединяются задания-задачи, которые ориентированы на выполнение работ по реализации документирования дипломного проекта, создание и наполнения заданий-прецедентов содержанием дипломной пояснительной записки.

Список литературы

1. **Сайт** сообщества Jazz. URL:http://jazz.net

2. **Fryer K., Gothe M.** Global Software Development and Delivery: Trends and Challenges URL:http://www.ibm.com/developerworks/rational/library/edge/08/jan08/fryer_gothe/ index.html

3. **Мельков А.** Средства автоматизации коллективной разработки программных проектов. URL: http://melkov.narod.ru

4. **Jazz** Integration Architecture Overview, [Электронный ресурс] URL:https://jazz.net/projects/DevelopmentItem.jsp?href=content/project/plans/jia-overview/index.html

5. **Meneely A., Williams L.** On Preparing Students for Distributed Software Development with a Synchronous, Collaborative Development Platform. URL: URL: http://www4.ncsu.edu/~apmeneel/p529-meneely.pdf

УДК 004.414.2

Т. Э. Галиев, аспирант,
Московский государственный институт электроники и математики
(технический университет),
e-mail: timur_galiev@mail.ru

Имитационные модели поиска информации в корпоративных поисковых системах

Рассматриваются основные процедуры организации поиска информации в корпоративных поисковых системах, приводятся математические и имитационные модели, описывающие поисковые процедуры и оценивающие число шагов до нахождения искомых данных и загрузку каналов связи.

Ключевые слова: *процедуры организации поиска, имитационные модели процедур поиска*

На сегодняшний день организации накопили внушительные объемы информации, как структурированной, так и неструктурированной. Данные хранятся на почтовых и Web-серверах, в электронных таблицах, в файлах приложений и баз данных. Компания может хранить сведения, касающиеся клиентов, сразу во многих источниках. Часто бывает, что данные представлены в разных форматах потому, что компании пользуются несколькими системами хранения, или потому, что они приобрели другие компании или слились с ними.

Создание интегрированных распределенных информационных систем (ИРИС), обеспечивающих доступ к данным, представляет собой качественно иной уровень производства, хранения, организации и распространения самой разнообразной информации (текст, графика, звук, видео и др.). Эффективная работа с ИРИС невозможна без наличия в ней механизмов поиска и ранжирования результатов.

Рассмотрим специализированную распределенную корпоративную систему и возможные варианты организации процедуры поиска в ней. Система состоит из поискового сервера и множества подсистем, образующих распределенную информационную среду. В общем случае систему можно описать следующим образом: на поисковый сервер поступают запросы от пользователей на поиск информации в различных подсистемах. Поисковый сервер, получив запрос от пользователя, в зависимости от имеющейся информации о подсистемах, посылает запросы в подсистемы. Подсистемы формируют ответ и посылают поисковому серверу. Сервер передает ответ пользователю.

Рассмотрим три наиболее часто встречающихся варианта организации поиска в корпоративных распределенных системах [1]: случайный поиск, поиск методом последовательного перебора, направленный поиск.

Случайный поиск. Поступивший от пользователя запрос обрабатывается сервером и поступает в случайно выбранную подсистему. Если информация найдена, то поиск прекращается, если нет — поиск продолжается до тех пор, пока не будет найдена информация. Каждая следующая подсистема выбирается случайно. Такой алгоритм поиска соответствует случаю, когда каждый пользователь самостоятельно разыскивает в системе требуемую информацию, предварительно не зная, где она находится.

Поиск методом последовательного перебора. Поступивший от пользователя запрос обрабатывается сервером и поступает в подсистемы в заданной последовательности. Поиск прекращается, как только информация найдена. В этом случае предполагается, что сервер собирает данные об информационных ресурсах и ранжирует их, составляя список в зависимости от вероятности нахождения искомого ресурса в той или иной подсистеме.

Направленный поиск с использованием метаданных. Поступивший от пользователя запрос обрабатывается сервером. На сервере однозначно определяется подсистема, которая содержит нужную информацию. Далее посылается запрос только в определенную подсистему. В этом случае поисковый сервер имеет полную информацию о содержимом подсистем, которая собирается по предварительным опросам. Организация такого метода поиска требует создания интегрированного банка метаданных (ИБМ). ИБМ — хранилище "данных о данных", содержащихся в подсистемах. Содержание ИБМ сопряжено со следующими издержками: требуется периодическое обновление, для уменьшения вероятности того, что в ответ на запрос пользователя будет получена неактуальная информация; процедура обновление ИБМ создает трафик, который загружает каналы передачи данных.

На практике в момент обновления ИБМ сервер не может обрабатывать пользовательские запросы (т. е. работает с отказами).

Анализ подобной системы требует построения соответствующих математических моделей, разработки общесистемных критериев качества для сравнения различных вариантов [2].

Важной характеристикой работы поисковых процедур является число шагов до поиска нужной информации. Зная эту характеристику, мы сможем с легкостью оценить среднее число поисковых запросов и загрузку каналов связи в ИРИС.

В результате построения математических моделей для каждого из трех приведенных выше методов организации

поиска были получены представленные формулы, оценивающие число шагов до поиска нужной информации.

Число шагов до нахождения требуемой информации

При процедуре случайного поиска число шагов до нахождения нужной информации (N_1) вычисляется по формуле:

$$N_1 = z_0(\sum_{r=0}^{M+1} G^r)e,$$

где z_0 — вектор-строка, компоненты которого есть вероятности начального состояния системы, M — количество подсистем, образующих ИРИС, e — единичный вектор-столбец, G — матрица вида:

$$G = \begin{pmatrix} 0 & (M-1)/M & 0 & 0 & \dots & 0 \\ 0 & 0 & (M-2)(M-1) & 0 & \dots & 0 \\ 0 & 0 & 0 & (M-3)(M-2) & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 1/2 \\ 0 & 0 & 0 & 0 & \dots & 0 \end{pmatrix}.$$

При поиске в заданном порядке среднее количество запросов (N_2), необходимое для нахождения нужной информации, вычисляется по формуле $N_2 = \sum_{i=1}^{M} ip_i$, где p_i — вероятность нахождения нужной информации в i-подсистеме.

При направленном поиске принципиальным отличием от приведенных выше способов организации поиска является отсутствие необходимости опрашивать каждую подсистему на наличие требуемой информации, так как поисковый сервер на основе имеющейся в ИБМ информации сам определяет такую подсистему. Поэтому число шагов до нахождения требуемой информации (N_3) заранее известно и равно 1.

Имитационные модели

Часто возможности математического моделирования комплексных систем ограниченны, поэтому мы прибегнем к имитационному моделированию. Имитационные модели реализуются в качестве аппаратных комплексов и программ для ЭВМ.

Для создания имитационных моделей поиска информации в ИРИС был выбран следующий инструментарий: язык программирования Visual C# и Microsoft Visual Studio 2008 .NET в качестве среды разработки. Язык Visual C# выбран не случайно: он прост и надежен (запрещена прямая манипуляция памятью, более строгие правила преобразования типов, запрет множественного наследования); является полностью объектно-ориентированным с мощными возможностями наследования и универсализации.

Разработанное программное обеспечение выполняет следующие задачи: моделирование процессов поиска в ИРИС; графический вывод результатов моделирования

При моделировании предметной области были выделены следующие основные сущности: поисковая система, подсистема, результат поиска, эксперимент, генератор случайных чисел, поисковый сервер. Для реализации имитационных моделей была разработана следующая модель классов:

Классы, представляющие ИРИС с различными вариантами организации поиска:

IntegratedSystemBase — базовый абстрактный класс, представляющий ИРИС; **IntegratedSystemWhithSpecifiedOrderSearch** — базовый класс, представляющий ИРИС с поиском «в заданном порядке»; **IntegratedSystemWhithSpecifiedOrderSearchProbabilitySection** — класс, представляющий ИРИС с поиском «в заданном порядке», при котором системы опрашиваются в порядке убывания вероятности нахождения информации в них; **IntegratedSystemWhithSpecifiedOrderSearchGoToRandom** — класс, представляющий ИРИС с поиском «в заданном порядке», при котором поиск начинается с опроса случайно выбранной подсистемы; **IntegratedSystemWhithDirectedSearch** — класс, представляющий ИРИС с «направленным поиском»; **IntegratedSystemWhithRandomSearch** — класс, представляющий ИРИС со «случайным поиском».

Классы, представляющие результаты поиска:

SearchResultBase — базовый абстрактный класс, представляющий результат поиска; **SearchResult** — класс, представляющий результат поиска; **AverageSearchResult** — класс, представляющий усредненный результат поиска.

Классы, представляющие эксперимент:

Expirement — базовый абстрактный класс, представляющий эксперимент; **ExperimentWhithDirectedSystem** — класс, представляющий эксперимент с системой с «направленным поиском»; **ExperimentWhithRandomSystem** — класс, представляющий эксперимент с системой со «случайным поиском»; **ExperimentWhithSpecifiedOrderSystem** — класс, представляющий эксперимент с системой с «поиском в заданном порядке».

Классы, представляющие другие сущности:

Subsystem — класс, представляющий подсистему (часть ИРИС); **SubsystemInfo** — класс, представляющий описание подсистемы в ИРИС; **OrderedSystemSearchType** — перечисление, представляющее тип алгоритма поиска в системе с «направленным поиском»; **RandomGenerator** — класс, представляющий генератор случайных чисел; **SearchServer** — класс, представляющий поисковый сервер.

Выводы

По результатам проведенных исследований можно сделать следующие выводы.

При процедуре случайного поиска не требуется дополнительной информации о вероятностях нахождения нужных данных в подсистемах, составляющих ИРИС. По сравнению с процедурой направленного поиска, объем пересылаемых данных при одинаковом количестве подсистем, составляющих ИРИС, больше примерно в два раза. Поэтому процедуру случайного поиска целесообразно применять только при отсутствии информации о содержании подсистем.

Направленный поиск при интенсивном потоке пользовательских поисковых запросов эффективней по сравнению с другими процедурами организации поиска, так как в этом случае загрузка каналов связи меньше, чем при использовании других процедур. Однако стоит отметить, что при таком способе организации поиска требуется предварительная подготовка, связанная с наполнением и обновлением ИБМ, что тоже загружает каналы связи. Сбор и обновление данных для ИБМ можно перенести на период наименьший загрузки каналов связи пользовательскими поисковыми запросами.

Список литературы

1. **Ганзер Х.** Введение в веб-сервисы. URL: // http://www.interface.ru/home.asp?artId=3281, (дата обращения: 10.12.2011)

2. **Вишневский В. М.** Теоретические основы проектирования компьютерных сетей. М.: Техносфера, 2003. 512 с.

3. **Моисеев Н.Н.** Математические задачи системного анализа. М.: Наука, 1981. 488 с.

CONTENTS

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

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

Сдано в набор 09.12.2011 г. Подписано в печать 25.01.2012 г. Формат 60×88 1/8. Заказ PI112.
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
105120, г. Москва, ул. Нижняя Сыромятническая, д. 5/7, стр. 2, офис 2.