2. Метапоисковая мультилингвистическая система поиска узкоспециализированной информации / И. Н. Карцан, П. В. Зеленков, Д. А. Рагзин и др. М., 2007. Зарег. во Всерос. научн.-техн. информ. центре, № 50200701673, рег. № ОФАП 8891.

3. Зеленков, П. В. Проблема развития метапоисковых технологий / П. В. Зеленков, Т. А. Ковалева // Вестник НИИ СУВПТ : сб. науч. тр.: НИИ систем упр., волновых процессов и технологий. Вып. 14. Красноярск, 2004. С. 95–103.

M. V. Karaseva, P. V. Zelenkov

REALIZATION OF THE DATA SEARCH MODULE ON THE BASES OF MULTILINGUAL THESAURUSES

The model of searching, ranking and relevance level of documents determination is introduced. It is done with the help of meta-search multilingual algorithms of data processing and control.

Keywords: ranking, relevance level, meta-search.

УДК 004.052.3

П. В. Ковалев, А. Н. Лайков, С. Н. Гриценко

ОПРЕДЕЛЕНИЕ НАДЕЖНОСТИ МУЛЬТИВЕРСИОННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ С ИСПОЛЬЗОВАНИЕМ МЕТОДОВ АНАЛИЗА СЕТЕЙ

Предложена методика определения надежности мультиверсионного программного обеспечения, которая позволяет использовать алгоритмы и методы анализа сетей для определения надежности программного обеспечения, разработанного с использованием мультиверсионного подхода.

Ключевые слова: надежность, мультиверсионность, сети, методы оценки, методы анализа сетей.

На заре компьютерной эры первостепенной задачей являлось развитие аппаратных компьютерных средств. В первую очередь это было обусловлено высокой стоимостью обработки и хранения данных. Однако впоследствии успехи микроэлектроники привели к резкому увеличению производительности компьютеров при значительном снижении их стоимости. поэтому основной задачей в 1990-х гг. и начале XXI в. стало совершенствование качества компьютерных приложений, возможности которых целиком определяются программным обеспечением (ПО).

Современный персональный компьютер теперь имеет производительность большой ЭВМ 1980-х гг. Сняты практически все аппаратные ограничения на решение задач, а оставшиеся ограничения приходятся на долю ПО.

Чрезвычайную актуальность приобрели следующие проблемы:

– аппаратная сложность опережает умение строить ПО, использующее потенциальные возможности аппаратуры;

– умение строить новые программы отстает от требований к новым программам;

– реализации возможностей по эксплуатации существующих программ угрожает низкое качество их разработки.

Ключом к решению этих проблем является грамотная организация процесса создания ПО и реализация технологических принципов промышленного конструирования программных систем (ПС) [1]. Кроме того, существует ряд способов повышения надежности программного обеспечения, одним из которых является введение избыточности.

Избыточность как метод повышения надежности ПО. За последние несколько лет тема мультиверсионного программного обеспечения не раз затрагивалась в различных диссертационных работах, например в [2]. Согласно этой работе, сбой программных систем может повлечь за собой большие потери и иметь весьма серьезные последствия. Поскольку абсолютная уверенность в безупречности программных средств достигается редко, то для выполнения требований к надежности проекта применяются методы повышения отказоустойчивости программного обеспечения. Программная отказоустойчивость достигается благодаря использованию алгоритмов программирования и методов разработки ПО, которые повышают вероятности того, что конечная реализация проекта приведет к правильному и (или) безопасному результату. Так как правильность и безопасность – концепции системного уровня, то потребность и степень использования программной отказоустойчивости непосредственно зависит от предназначения приложения и полного проектирования систем [2].

Избыточность, применяемая для обеспечения надежности функционирования комплекса программ (КП), используется прежде всего для контроля и селекции искажений вычислительного процесса или данных и для выработки мер по снижению последствий этих аномалий. Основная задача состоит в ограничении или исключении возможности аварийных последствий, соответствующих отказу системы в процессе функционирования.

Под временной избыточностью понимается использование некоторой части производительности ЭВМ для контроля исполнения программ и восстановления вычислительного процесса. Для этого при проектировании программ должен предусматриваться запас производительности, который затем будет применяться для контроля и повышения надежности функционирования ПО. Диагностика искажений и операций восстановления в общем случае требует небольшого интервала времени, который выделяется либо за счет резерва, либо за счет сокращения времени решения функциональных задач.

Информационная избыточность состоит в дублировании накопленных исходных и промежуточных данных, обрабатываемых КП. Эта избыточность используется для сохранения достоверности данных, которые в наибольшей степени влияют на нормальное функционирование программ или требуют значительного времени для восстановления. Информационная избыточность может способствовать не только обнаружению искажений данных, но и устранению ошибок. Для этого данные защищают двух- или трехкратным дублированием с соответствующей дисциплиной контроля сохранности и периодическим обновлением.

Программная избыточность служит для контроля и обеспечения достоверности наиболее важных результатов обработки информации. Она заключается в применении в КП нескольких вариантов программ, различающихся методами решения некоторой задачи или программной реализацией одного и того же метода. Программная избыточность также необходима для реализации программ контроля и оперативного восстановления данных с использованием информационной избыточности и для функционирования всех средств защиты, использующих временную избыточность [3].

Одним из наиболее перспективных методов программной избыточности является метод мультиверсионного проектирования.

Применение методологии мультиверсий для обеспечения отказоустойчивости программного обеспечения. Мультиверсионная отказоустойчивость основана на использовании двух или более версий, или вариантов, модуля программного обеспечения, исполняемых последовательно или параллельно. Версии используются как альтернативы (с отдельными средствами обнаружения ошибок), в парах (чтобы осуществить обнаружение проверками дублирования) или в больших группах (чтобы маскировать ошибки через голосование). Использование множественных версий обосновывается предположением о том, что по-разному построенные компоненты, т. е. компоненты, построенные различными проектировщиками, различными алгоритмами, различными инструментальными средствами проектирования и т. д., должны иметь разные ошибки [4]. Поэтому если одна версия производит сбой на специфическом вводе, то по крайней мере одна из альтернативных версий должна обеспечить корректный вывод. Рассмотрим далее краткие описания некоторых из этих методов, применяемых для повышения надежности и безопасности программного обеспечения.

Блоки восстановления. Методика блоков восстановления [5; 6] объединяет основные идеи метода контрольных точек и рестарта для мультиверсионных компонентов программного обеспечения таким образом, что различные версии используются только после того, как обнаруживается ошибка (рис. 1).

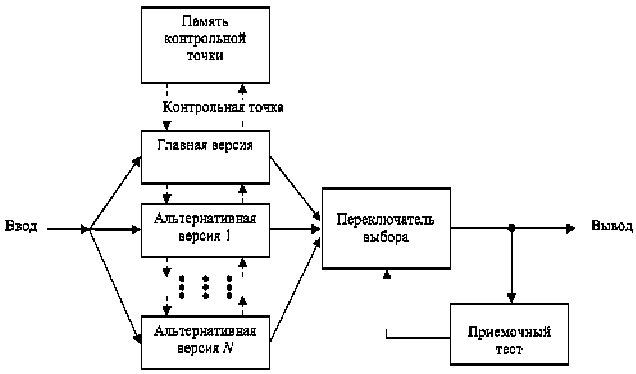

Рис. 1. Модель блока восстановления

Контрольные точки определяются перед исполнением версий. Эти точки необходимы, чтобы восстановить состояние после того, как версия произведет сбой и не сможет обеспечить корректную начальную точку для следующего компонента.

Приемочный тест не должен быть только выходным тестом и может осуществляться различными встроенными проверками, чтобы увеличить эффективность обнаружения ошибок. Что касается альтернативных версий, то они могут быть разработаны с более низкой производительностью и качеством, например за счет вычисления значений с меньшей точностью, из-за того, что первичная версия будет выполняться успешно в большинстве случаев. Подобно методу разнообразия данных, вывод альтернативных версий может быть разработан таким образом, чтобы быть эквивалентным первичному с определением эквивалентности, зависящей от приложения.

Фактическое выполнение множественных версий может быть последовательным или параллельным в зависимости от вычислительных мощностей и требований производительности. Если все альтернативы выдали сбой, то компонент должен инициировать исключение, чтобы сообщить остальной части системы об отказе завершения его функции. Следует отметить, что такое возникновение отказа не подразумевает постоянного отказа компонента и он может быть повторно использован после изменения его состояния или ввода. Возможность совпадения отказов является источником серьезных дискуссий относительно всей технологии отказоустойчивого мультиверсионного программного обеспечения.

***N*-версионное программирование.** *N*-версионное программирование [7] – это мультиверсионная методика, в которой все версии разработаны в соответствии с идентичными основными требованиями, а решение о правильности вывода основано на сравнении всех выводов (рис. 2).

Использование универсального алгоритма выбора решения (обычно голосования) для выбора правильного вывода составляет фундаментальное отличие этого подхода от подхода с блоком восстановления, который требует зависимого от приложения приемочного теста. Так как все версии построены в соответствии с идентичными

требованиями, то применение *N*-версионного программирования требует значительно больших затрат при разработке, но эта сложность, т. е. трудность разработки, не обязательно будет намного больше, чем сложность формирования одной версии. Проектирование выбирающего алгоритма может быть усложнено необходимостью реализовать алгоритм неточного голосования. Многие исследования направлены на разработку методологий, которые увеличивают вероятность достижения эффективного разнообразия в конечном продукте.

Фактическое выполнение версий может быть последовательным или параллельным. Последовательное выполнение может потребовать использования контрольных точек, чтобы перезагрузить состояние перед выполнением дополнительной версии.

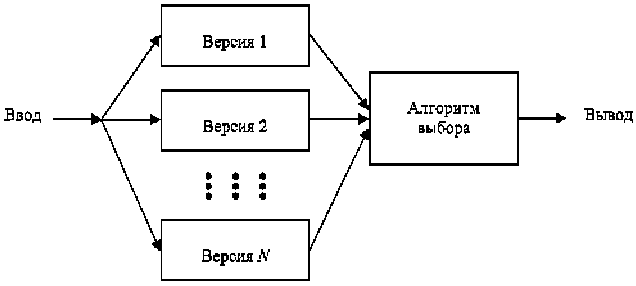

Рис. 2. Модель *N*-версионного программирования

Блоки восстановления с согласованием. Блоки восстановления с согласованием [8] – это подход, комбинирующий *N*-версионное программирование и метод блоков восстановления для повышения надежности по сравнению с использованием только одного из подходов. Согласно [8], приемочные тесты в блоках восстановления страдают из-за недостатка руководящих принципов для их разработки и общей склонности к ошибкам проектирования из-за трудности создания эффективных тестов.

Использование голосования, как в *N*-версионном программировании, не может быть применено во всех ситуациях, особенно когда возможны правильные множественные выводы. В этом случае голосование объявило бы об отказе при выборе соответствующего вывода. Блоки восстановления с согласованием используют алгоритм выбора решения, подобный *N*-версионному программированию, как первый уровень решения. Если этот уровень объявляет отказ, то используется второй уровень приемочного испытания, аналогичный тому, который был использован в методе блоков восстановления. Реализация этого подхода является гораздо более сложной, чем реализация любого индивидуального метода, однако сравнение модели надежности показывает, что этот комбинированный подход имеет высокий потенциал для создания более надежного программного обеспечения [8]. Использование слова *потенциал* здесь важно потому, что добавленная сложность могла бы фактически работать против проекта и приводить к менее надежной системе.

Существуют и другие методы построения мультиверсионного ПО, в том числе основанные на модификации тех методов, которые были описаны выше. Рассмотрим далее возможность применения методов анализа сетей для анализа надежности мультиверсионного ПО.

Использование методов анализа сетей для определения надежности мультиверсионного ПО. Прежде всего необходимо отметить, что методы анализа сетей уже были успешно использованы Д. М. Письманом для исследования характеристик работы узлов распределенных систем обработки информации [9]. Одной из ключевых особенностей описываемых им GERT-сетей является возможность их использования в случаях, когда применение методов, основанных на статических или детерминированных моделях систем или программ, невозможно [9].

Современное общество отчасти можно рассматривать как систему сетей, предназначенных для транспортирования, передачи и распределения электроэнергии, товаров и информации. Поскольку каждая из подсистем этой системы имеет весьма сложную структуру и является дорогостоящей, то возникает необходимость в эффективном использовании уже существующих технических средств и в рациональном проектировании новых.

При разработке и совершенствовании больших и сложных систем, а также при поиске путей их наиболее рационального использования существенное значение приобретают методы сетевого анализа. В первую очередь это связано с тем, что с помощью сетей можно довольно просто построить модель системы. Кроме того, методы сетевого анализа позволяют построить модель сложной системы как совокупности простых систем; составить формальные процедуры для определения качественных характеристик системы; указать механизм взаимодействия компонентов управляющей системы для описания последней в терминах ее основных характеристик; определить, какие данные необходимы для исследования системы; провести начальные исследования управляющей системы и составить предварительное расписание работы ее компонентов.

Основное достоинство сетевого подхода заключается в том, что он может быть успешно применен к решению практически любой задачи, если исследователь обладает необходимыми знаниями и способностью точно построить сетевую модель. Преимущества использования сетевых моделей можно сформулировать следующим образом:

– сетевые модели могут точно описать многие реально существующие системы;

– сетевые модели являются несложными для пользователей, не имеющих специального математического образования и связанных, как правило, с решением конкретных прикладных задач;

– сетевые алгоритмы позволяют находить наиболее эффективные решения при изучении некоторых больших систем;

– по сравнению с другими методами оптимизации сетевые алгоритмы позволяют решать задачи со значительно большим числом переменных и ограничений. Это становится возможным благодаря тому, что при использовании методов сетевого анализа часто удается ограничиться изучением лишь части рассматриваемой системы [10].

Любой комплекс программ является сложной системой, а значит обладает всеми свойствами сложных сис-

тем, включая возможность расчленения на группы наиболее тесно взаимодействующих элементов – подсистемы, имеющие свое специальное назначение и цель функционирования [3]. Это в свою очередь означает, что любой комплекс программ может быть представлен в виде сети, а следовательно, для его расчета применимы методы анализа сетей.

Введем некоторые определения. Пусть направленный граф G состоит из непустого множества E направленных ребер (дуг) и непустого множества V вершин (узлов). В дальнейшем пары слов *ребро* и *дуга*, *вершина* и *узел* будем считать синонимами. Пусть каждое ребро однозначно определяется по его начальному i и конечному j узлам.

Будем называть *маршрутом* последовательность дуг, в которой конечный узел одной дуги является начальным узлом следующей дуги.

Направленный взвешенный связанный граф будем называть *сетью*, при этом каждый узел сети активируется с некоторой вероятностью.

Выполнением сети будем называть процесс выполнения случайного эксперимента, в то время как *реализацией сети* будем называть итог данного эксперимента.

Весом дуги $<i,j>$ для стохастической GERT-сети является вектор $[p_{ij}, F_{ij}]$, где p_{ij} – условная вероятность выполнения дуги $<i, j>$ при условии активации узла i (более кратко – вероятность выполнения дуги (работы) $<i, j>$); F_{ij} – условная функция распределения времени выполнения дуги $<i, j>$ при условии, что $<i, j>$ выполняется.

Каждый узел сети имеет входную и выходную функции активации, также влияющие на параметры активируемого узла.

Существуют следующие виды входных функций:

– AND-функция (узел активируется, если выполнены все дуги, входящие в него);

– IOR-функция (узел активируется, если выполнена любая дуга, входящая в него);

– EOR-функция (узел активируется, если выполнена любая дуга, входящая в него, при условии, что в данный момент времени может выполняться только одна дуга, входящая в данный узел).

Выходные функции бывают *детерминированными* (все дуги, выходящие из узла, выполняются, если узел активирован) и *стохастическими* (только одна дуга, выходящая из узла, выполняется с заданной вероятностью, если узел активирован).

Комбинируя все входные и выходные функции (рис. 3), получаем шесть различных типов узлов.

Активация узла означает, что система перешла в некоторое состояние и определяет множество возможных дальнейших действий. Одно или несколько действий начинают свое выполнение сразу после активации узла, являющегося их началом. Активация узла происходит, если его входная функция выполнена. После выполнения выходной функции активированного узла, т. е. начала выполнения соответствующей дуги, он становится неактивным.

Таким образом, GERT-сеть – это сеть с источниками R и стоками S вида «работа на дуге», в которой каждый узел принадлежит одному из шести типов узлов, для каждой дуги $<i, j>$ определен вес вида $[p_{ij}, F_{ij}]$ с указанными выше значениями и задано начальное распределение источников сети.

Например, модель N-версионного программирования (см. рис. 2) может быть представлена в виде GERT-сети, каждый узел которой состоит из EOR-входа и стохастического выхода (рис. 4). Если для каждого узла этой сети задать вектор $[p_{ij}, F_{ij}]$, то это позволит получить качественные характеристики системы.

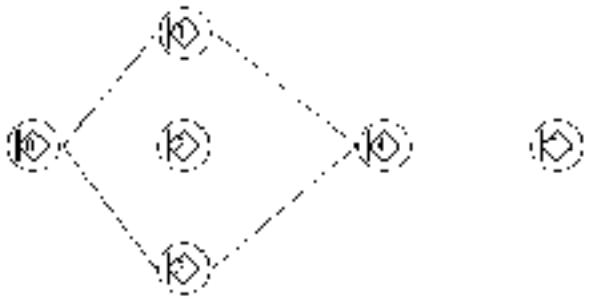

Рис. 4. Модель N-версионного программирования в виде GERT-сети

Итак, в заключение сделаем несколько выводов:

– повышение надежности программного обеспечения является актуальной задачей;

– избыточность (и ее более частный случай – мультиверсионность) как метод повышения надежности ПО является распространенным, перспективным и достаточно эффективным методом;

– методы сетевого анализа (в том числе частный случай сетей – GERT-сети) позволяют легко построить модель системы и составить процедуры для определения ее качественных характеристик;

– любой комплекс программ и все программное обеспечение как совокупность всех комплексов программ являются сложными системами и в соответствии с этим они могут быть подвергнуты декомпозиции и представлены в виде множества узлов и дуг или просто сети. Следовательно, любое мультиверсионное программное обеспечение также может быть представлено в виде сети;

– учитывая свойства методов анализа сетей, которые позволяют составить формальные процедуры для определения качественных характеристик системы, можно с большой степенью вероятности предположить, что надежность мультиверсионного программного обеспечения как одна из его качественных характеристик может быть рассчитана с использованием методов анализа сетей.

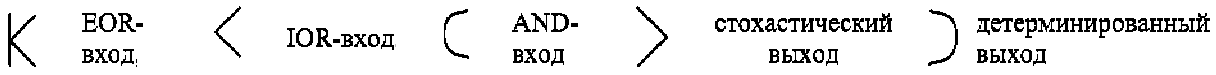

Рис. 3. Графическое обозначение входных и выходных функций GERT-сети

Библиографический список

1. Орлов, С. Технологии разработки программного обеспечения / С. Орлов. СПб. : Питер, 2002.

2. Поздняков, Д. А. Компонентная программная архитектура мультиверсионных систем обработки информации и управления : автореф. дис. … канд. техн. наук / Д. А. Поздняков. Красноярск, 2006.

3. Липаев, В. В. Проектирование программных средств : учеб. пособие / В. В. Липаев. М. : Высш. шк., 1990.

4. Avizienis, A. On the Implementation of N-Version Programming for Software Fault Tolerance During Execution / A. Avizienis, L. Chen // Proc. of the IEEE COMPSAC'77. Chicago, USA, 1977. P. 149–155.

5. Randell, B. The Evolution of the Recovery Block Concept / B. Randell, J. Xu // Software Fault Tolerance / ed. M. R. Lyu. New York : John Wiley&Sons, 1995. P. 1–21.

6. Predictably Dependable Computing Systems / ed. B. Randell [et al.]. Berlin : Springer-Verlag, 1995.

7. The Methodology of N-Version Programming / A. Avizienis // Software Fault Tolerance / ed. M. R. Lyu. New York : John Wiley & Sons, 1995. P. 22–46.

8. Scott, R. K. Fault-Tolerant Software Reliability Modeling / R. K. Scott, J. W. Gault, D. F. McAllister // IEEE Transactions on Software Engineering. 1987. Vol. SE-13, № 5. P. 582–592.

9. Письман, Д. М. GERT-сетевой анализ временных характеристик работы узлов распределенных систем обработки информации : автореф. дис. … канд. техн. наук / Д. М. Письман. Красноярск, 2006.

10. Филлипс, Д. Методы анализа сетей / Д. Филлипс, А. Гарсиа-Диас. М. : Мир, 1984.

P. V. Kovalev, A. N. Laykov, S. N. Gritsenko

THE RELIABILITY RESEARCH OF N-VERSION SOFTWARE USING METHODS OF NETWORK ANALYSIS

The technique of the reliability research of N-version software is offered. It allows using algorithms and methods of network analysis for the reliability research of software, developed by N-version approach using.

Keywords: reliability, N-version, networks, estimation methods, network analysis.

УДК 004.421.4

К. В. Бадмаева

АЛГОРИТМ ОЦЕНКИ РЕЛЕВАНТНОСТИ ПРЕДСТАВЛЕНИЙ ДЛЯ МАТЕРИАЛИЗАЦИИ В СПЕЦИАЛИЗИРОВАННОМ ХРАНИЛИЩЕ ДАННЫХ*

Рассмотрено проектирование специализированных хранилищ данных при отсутствии статистической информации о работе базы данных. Предложен алгоритм оценки релевантности представлений на основе данных предметной области, который позволяет принимать решение о включении в схему хранилища агрегированных данных. Алгоритм предназначен для уменьшения субъективности проектировщиков при создании эффективной первоначальной модели данных хранилища.

Ключевые слова: хранилище данных, материализация представлений.

Одной из наиболее важных задач проектирования информационных хранилищ данных является задача выбора представлений для материализации. Материализованными представлениями называются собранные в базе данных (БД) сводные таблицы, полученные выбором и агрегированием данных из других таблиц. Существующие алгоритмы выбора представлений основываются на статистике, собранной по полям таблиц хранилища сервером БД в процессе эксплуатации системы [1–4]. Поэтому разработка методов, способствующих повышению производительности хранилища на ранних стадиях проектирования, когда статистика еще отсутствует, актуальна и востребованна.

При проектировании хранилищ необходимо решить проблему выбора представлений, которые наиболее целесообразно включить в схему данных при наличии ограничений на системные ресурсы. В общем случае задача выбора относится к классу NP-трудных [5], и для ее решения применяются эвристические методы.

Автором данной статьи предлагается оценивать релевантность, т. е. важность с точки зрения пользователей хранилища, представлений на основе исследования предметной области, определения целей, задач и наборов методов анализа данных еще до того, как в базе данных будет накоплена достаточная статистика. Определение ре-

* Работа выполнена при поддержке гранта Президента РФ для ведущих научных школ № НШ-3431.2008.9.