COMPUTER SCIENCE

ОБНАРУЖЕНИЕ ОШИБОК ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПОСРЕДСТВОМ МЕТОДА РЕВЕРСНОГО ПОВТОРЕНИЯ

доцент, PhD Христова М. П.

Болгария, София, Транспортны Университет имени Тодор Каблешков

Abstract. *There is a set of admissibility requirements on the erroneous effects of the managed object established upon the computer systems for control over the responsible technological processes in the transportation, aviation, space, energy, medicine and other fields. These requirements are primarily related to faultlessness of the software. One way to achieve this is through a multi-version programming and a comparison of the results of the used versions. Two versions of the diversity method are most often used for the detection and subsequent removal of errors in the software. The aim should be for both versions to be maximally independent. It is neither always possible nor easy to find at least two methods for solving the problem (algebraically, logical, technical) on which to build algorithms. The proposed method of reverse iteration offers a solution to a direct problem in the first version and reverse problem in the second version. The reversion solves the problem in the diversity of the software within the second method, which otherwise would be impossible or very difficult. Moreover, it offers a solution via a universal approach by offering radically different methods. The article illustrates the method both as a principle, as well as during implementation on a computer base. An analysis is conducted on the safety compared with the classical scheme of "two by two" when solving two direct problems in different versions. It is established that the method has certain advantages. Except in conditions of greater safety from failures, they are in greater efficiency. It is deemed unnecessary to have two different programming teams for the software, which is a costly solution. One programmer is sufficient and can build both versions.*

Keywords: *Safety critical software; Safety critical Systems; Software error; Diversity; Dual channel Systems.*

1. ПОСТАНОВКА ПРОБЛЕМЫ

На транспорте, в медицине, в энергетике, в авиации и космосе, а также и в ряду других областей функционируют особо ответственные технологические процессы (ООТП). Выход ООТП из штатного функционального состояния может привести к гибели людей и большим материальным потерям и/или к недопустимому ущербу окружающей среде. Управляет ООТП один из классов систем реального времени. Ниже эти системы коротко обозначены как критические по безопасности системы - КБС (Safety Critical Systems) [1]. Управление воздушным движением, скоростными поездами, реакторами атомной энергетики, оборудованием для хирургических операций и т.д. являются весьма ответственным технологическим процессом, связанным с сохранением жизни людей и сбережением крупных материальных ресурсов. Невыполнение функций КБС угрожает жизни людей. Поэтому к ним предъявляют особо высокие требования.

Ныне КБС выполнены на компьютерной (микропроцессорной) базе, а требования к ним распространяются на программное обеспечение – критический софтвер. Критическим софтвером достигают:

1. заданную функциональность,
2. требуемую безопасность.

Функциональность является характеристикой любой системы, в том числе системы без претензий на безопасность. Для того, чтобы заставить микропроцессор работать как бортовой котроллер скорости поезда, как чёрный ящик самолёта или как охранное устройство необходимо его программировать соответствующим образом. Это дело специалистов по отраслям и не является предметом этой статьи.

Вторая характеристика – **безопасность** - весьма специфична и относится только в классу КБС. К критическому софтверу ставят повышенные требования к надежности и безопасности в смысле недопустимости ошибочных управляющих воздействий на объект управления [2]. Цена отказа критической по безопасности системы так высока, что для разработки, проектирования и внедрения таких систем, стоит применят более ресурсоемкие методы, которые для других систем экономически невыгодны. Один из известных методов достижения высокой надёжности – много (n-) версионное программирование [3]. В аппаратном выполнении оно соответствует обработки информации многоканальной системой из устройств (каналов) одинакового предназначения и выбор выходного сигнала по критерию верности, например два из трёх.

Анализ известного из специализированной научной литературы показывает, что проблемы критического софтвера охватывают формальные методы синтеза и анализа, безопасное программное обеспечение, сопровождение софтвера на всем протяжении его жизненного цикла, языки программирования, контроль и сертификацию программного обеспечения, надежность и безопасность системы с критическим софтвером и др. Но несомненно доминирует тема о ошибках. Ошибки допускают на всех этапах: спецификация, разработка, проектирование, кодирование программы сопровождение софтвера и документация. Исследуют непреднамеренные, случайные ошибки, причины ошибок, место ошибок, ошибки персонала, последствия ошибок и т.д.

В отличии от аппаратного компонента, где исправности одного экземпляра устройства не имеют никакое отношение к исправности других, то софтверные ошибки являются неотъемлемый дефект всех устройств данной серии, работающие по данной программе. Если копия программы работают во всех устройств многоканальной системы, ошибки, мультиплицированные по всем каналам, сравнением на выходе невозможно обнаружит.

Как обнаружит ошибки в софтвере при тестировании (off-line) или во время эксплуатации (on-line), для того, чтобы устранить их своевременно? Это один из ключевых вопросов, чему посвящается и это статья. Вопросы преднамеренных злоумышленных ошибок не рассматриваются.

2. DIVERSITY (ДИВЕРСИТЕТ)

Один из признанных методов решения задачи обнаружения ошибок является диверситет (Diversity) [2].

Diversity – *это метод решения задачи (математической, логической, технической и т.д.) двумя (A и B) способами при одинаковых входных данных.*

Исходят из того, что по крайней мере есть два различных решения данной проблемы, хотя и это не всегда справедливо. Критерием верностью решения задачи в диверситетной системе является соответствие (в частном случае - равенство) двух полученных *A и B* результатов. Ошибки, в том числе в принципах построения и алгоритмах работы обоих каналов, могут быть обнаружены путем сравнения выходных результатов.

Примером рассматриваемого диверситетного метода является решение квадратного уравнения. Пусть программа А решает уравнение по классической формуле:

$$r_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \qquad (1)$$

Программа В решает задачу по формулам Виета:

$$r_1 + r_2 = \frac{b}{a}$$
$$r_1 r_2 = -\frac{c}{a} \qquad (2)$$

Получение результаты должны быть одинаковыми. Если они не совпадают, значить есть ошибка.

Принципиально ошибки могут находиться в сорсе, в алгоритме, в методе решения, да и ещё выше – в концепции. При некоторых выходных данных и внутренних состояниях ошибки могут не проявляться, они остаются латентными, скрытыми. Но если софтвер построен без избыточностью, то в хотя бы одном состоянии они обязательно обнаруживаются. Для примера доказано, что математические методы (1) и (2) корректны. Значить, ошибка в алгоритмах или в сорсе.

Для применения метода диверситета необходимо программировать две версии решения задачи – А и В [3]. Не рекомендуется их возлагать одному коллективу программистов, поскольку в обоих версиях он может допустить одни и те же ошибки. Если они приведут к некорректными, но соответственными результатами, ошибки отстанут необнаруженными, а возможное воздействие на выходе – опасным.

3. МЕТОД РЕВЕРСНОГО ПОВТОРЕНИЯ

3.1. Основание предложения

Существуют методы диагностики, контроля и тестирования (абсолютное, относительное), обнаруживающих ошибок софтвера [ ]. Большинство этих методов и их практических применений связано с диверситетом в двухканальном исполнении и с относительным тестом «есть – есть» на выходе по принципу «два из двух». Тестировании может произойти of-line, или во время работы (on-line).

Не всегда возможно найти, по крайней мере два, эффективных методов решения задачи. Поэтому оба коллектива (А и В) программистов программируют свои версии по одинаковым алгоритмам, но в разных сорсах. В таких случаях обнаружить алгоритмические ошибки невозможно. Идентифицируют только программные ошибки (ошибки кодирования, сорса). Кроме того если программисты из одной школы, то они могут допустить одинаковые ошибки, которые приведет к одному и тому же результату. При сравнении на выходе ошибки не обнаружить, а выходное воздействие системы ошибочное и возможно опасное. Вероятность этого события и определяется количественная мера безопасности метода.

Предлагается **метод реверсного повторения** (МРП), который в принципе известен. В [4] метод иллюстрирован схемой, здесь показанной на рис. 1. Дальше метод разработан более детально.

Этим методом пытаются устранить недостатки формулы «два из двух». Не приходится искать два принципиально разные алгоритмы, чтобы обеспечить возможность обнаружения их ошибок при программировании. Эта возможность заложена в суть самого метода.

Предпосылкой МРП является возможность по выбранному методу решить «прямой» и «обратной» (реверсной) задачи. Простым примером является алгебраическая задача:

$$y = x^2 + 1 \tag{3}$$

Реверсная задача:

$$x = \sqrt{y - 1}\,. \tag{4}$$

Конечно, в больших системах не так это просто, как в (3) и (4), но принцип остается в силе. Входные данные для решения задачи управления запоминают и подают их для выполнения программой А. Результат решения А запоминают и подают его как входные данные для решения реверсной задачи В. Результат В сравнивают с входными данными А. Сравниваемые векторы должны совпадать. Если согласия есть, то софтвер даёт разрешение ОК, если нет, то в софтвере допущена ошибка. ОК снимается.

Критический по безопасности софтвер должен быть безошибочным. Если в данном примере это является программой бортового компьютера локомотива для вычисления скорости движения поезда в соответствии с параболой торможения, то ошибка может привести к скорости, большей, чем допустимой по безопасности. Поэтому «поймать» ошибку вопрос большой важности.

Рис. 1 Схема метода реверсного повторения

4. ПРИМЕНЕНИЕ МЕТОДА В КОМПЬЮТЕРНЫХ СИСТЕМАХ БЕЗОПАСНОСТИ

4.1. Блочная схема

В цифровых схемах, какими являются и микропроцессорные устройства, на входе А схемы поступают кодовые векторы X(x_1 x_2…, x_w) длиной *w* битов. В результате выходит вектор А Y(y_1 y_2…, y_v) длиной *v* битов (Рис.2). После обработки информационного потока формируют управляющий сигнал в виде комбинации (вектора) единиц и нулей.

Управляющим является выходной вектор, полученный по программе A, но под условием соответствия при сравнении.

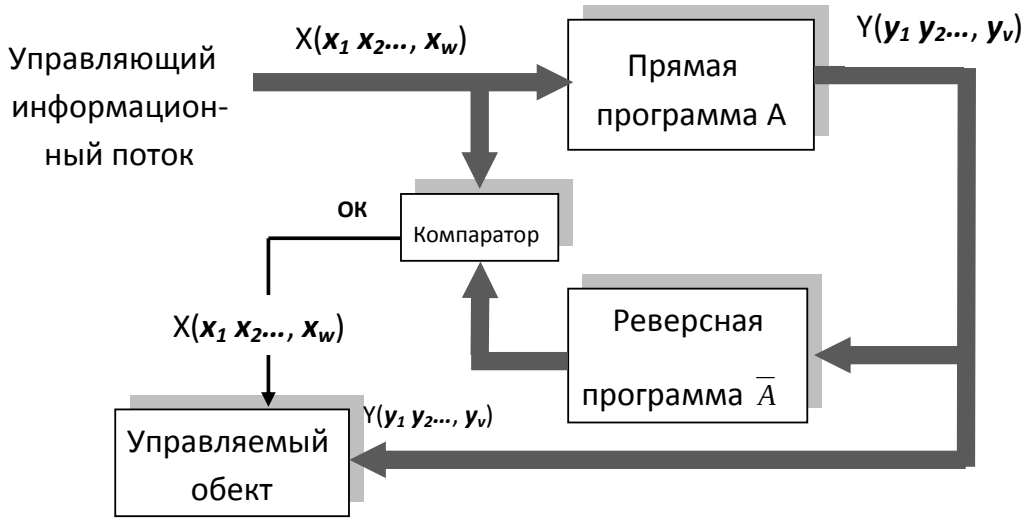

Рис. 2 Блок-схема метода реверсного повторения

Компаратор (сравнивающее устройство) обычно выполнен на софтвере. В случае эквивалентности, в частном случае идентичности, от даёт ОК. Управляемый объект получает право принять управляющий сигнал Y(y_1 y_2…, y_v). Обычно ОК включает электропитание к объекту.

Обработка информации по двум программам А и В может осуществиться последовательно в одном микропроцессорном устройстве (1 Hardwar и 2 Software - 1H+2S), но может и в двух разных устройствах (2H+2S).

4.2 Анализ безопасности

Безопасность метода *реверсного повторения* опирается на предпосылках:

- Активация ошибки в программе А вызовет некорректный выходной вектор А', который в реверсной программе приведет к неправильному конечному результату В'. При сравнении ошибка будет обнаружена.
- Отказ в реверсной программе вызовет некорректный конечный результат и опять будет обнаружена.
- Ошибки в программах А и В приведут к другому В'' несоответствующему результату и компаратор идентифицирует ошибки.
- Если по случайности есть ошибки, которые инверсны таким образом, что ошибка, искажающая вектора В'→В, компенсирует искажение вектора А→А' второй ошибкой, то они останутся не обнаруженными. Вероятность этого события и определить вероятность опасного отказа, поскольку выходное воздействие программой А на объект не будет снято, а и неизвестно к чему оно приведёт.

В этом отношении безопасность, по принципу, одинакова с безопасностью в системах «два из двух» с двумя прямыми версиями [5]. И в классическом двуверсионном решении существует возможность допущения ошибок в обеих программах, ведущих к некорректными неидентифицируемыми результатами.

Преимущества *метода реверсного повторения* состоит в том, что метод:

- Введя реверсную программу, создает естественным путем алгоритмический диверситет, чем преодолевает трудность нахождения два эффективные метода решения задачи;
- При решении прямой задачи в двух версии вероятность допущения неконтролируемых ошибок больше.
- Нет необходимости формировать две команды программистов, поскольку прямая и реверсная задача настолько разные, что их может программировать и один программист без опасности допустить неконтролируемых одинаковых ошибок.

Для того, чтобы рассматриваемая система была безопасной, необходимо безошибочная программа сравнения. Поскольку задача сравнения довольно проста, не трудно доказать её безошибочность.

4.2 Ограничения метода

Не всегда данный метод реверсного повторения применим. Принципиально, для применения метода необходимо однозначное соответствие между входом и выходом микропроцессорной обработки. Это условие в силу для всех функций одной переменной $y = f(x)$. Есть много задач, отвечающие этому условию. Могут быть довольно сложными программными задачами и простыми комбинационными автоматами как двоично-десятичные шифраторы/дешифраторы. Данному вектору входа соответствует однозначно определённый вектор выхода, и обратно. Например вектору 0101 соответствует число 5. И обратно.

Но в большинстве случаев нет такова однозначного соответствия, отчего прикладное пространство метода сужается.

Расширение прикладного пространства метода реверсного повторения возможно, если условие однозначности в данном случае необязательно Например, контроль только ответственных действиях как открытие сигнала светофора или положение стрелки в системах железнодорожной автоматики. Если условия безопасного движения по маршруту: стрелки в правильном положении, пути свободные от подвижного состава, нет враждебных маршрутов и т.д., сигнал можно открыть. Но этого не происходит, пока не решить реверсную задачу – при логической единице на выходе, посмотреть обратно, существуют ли все условия, при которых сигнал может открыть. Если решение обратной задачи подтвердит соответствие, то софтвер даёт ОК и открытие происходит.

В этом примере и в других подобных случаях нет необходимости решать реверсные задачи, если они не связаны с ответственным действием. Таким образом, подходя гибко и конкретно, можно повысить эффективность применения метода.

ЗАКЛЮЧЕНИЕ

Диверситет является одним из немногих способов обнаружения ошибок софтвера. Для того, чтобы обнаружить ошибки на всех уровнях решения задачи, необходимо различие версий

не только в кодировании, а и на уровне, на котором допущены и отыскивать ошибки – спецификация, методы и алгоритмы программ. Диверситет, и в частности его проявлении в предлагаемом методе, отличается универсальностью по отношении места ошибок и, по правило, он в состояние их обнаружить.

Применение диверситета для обнаружения ошибок в большей степени зависит от того, как найти **минимум двух** способов (принципов и методов) решения логической, алгебраической, технической, софтверной задачи. Предлагаемый метод прямого и реверсного решения и вывод результата после сравнения решает коренным образом эту проблему. Его эффективность опирается на двух преимуществах:

- меньшая вероятность допущения коррелированных ошибок в обоих версиях;
- возможность исключением второго состава программистов.

ЛИТЕРАТУРА

1. Knight J. C., Software Engineering, ICSE 2002, Proceedings of the 24rd International Conference on IEEE, 2002, ISBN:1-58113-472-X

2. EN50128 " Software for control and protection systems"

3. Ковалев И.В. Проблемы программной реализации мультиверсионной среды исполнения алгоритмов обработки информации в системах управления. Вестник Сибирского государственного аэрокосмического университета им. академика М.Ф. Решетнева, Выпуск № 4 (56), 2014

4. Hristov, H., V. Trifonov. Reliability and Security of the Communication. Novi Znania, Sofia, 2007

5. Hristov, H., Wang Bo. Safety Critical C**omputer** Systems: failure Independence and software diversity effects on Reliability of dual channel Structures. Information Technologies and Control. No 2, 2014, pp.9-19