УДК 658.5.012.7

Вестник СибГАУ
2014. № 4(56). С. 62–73

ПРОБЛЕМЫ ПРОГРАММНОЙ РЕАЛИЗАЦИИ МУЛЬТИВЕРСИОННОЙ СРЕДЫ ИСПОЛНЕНИЯ АЛГОРИТМОВ ОБРАБОТКИ ИНФОРМАЦИИ В СИСТЕМАХ УПРАВЛЕНИЯ

И. В. Ковалев

Сибирский государственный аэрокосмический университет имени академика М. Ф. Решетнева
Российская Федерация, 660014, г. Красноярск, просп. им. газ. «Красноярский рабочий», 31
E-mail: kovalev.fsu@mail.ru

Рассматривается проблема программной реализации мультиверсионной среды исполнения алгоритмов обработки данных в информационно-управляющих системах. Показана возможность использования существующих мультиверсионных моделей повышения надежности программного обеспечения для реализации среды исполнения алгоритмов обработки информации в системах управления. Разработанный программный инструментарий позволяет унифицировать применение мультиверсионного подхода к различным программным комплексам и, в отличие от существующих в настоящее время, исполнять модули не только с помощью методологии мультиверсионного программирования, но также с использованием наиболее распространенных мультиверсионных моделей: восстанавливающихся блоков, согласованных восстанавливающихся блоков, t/(n–1)–версионного программирования и мультиверсионного программирования с самопроверкой.

Ключевые слова: обработка информации, мультиверсионное программное обеспечение, система управления, отказоустойчивость.

Vestnik SibGAU
2014, No. 4(56), P. 62–73

PROBLEMS OF SOFTWARE IMPLEMENTATION OF MULTIVERSIONED VIEWS RUNTIME OF ALGORITHMS FOR DATA PROCESSING IN CONTROL SYSTEMS

I. V. Kovalev

Siberian State Aerospace University named after academician M. F. Reshetnev
31, Krasnoyarsky Rabochy Av., Krasnoyarsk, 660014, Russian Federation
E-mail: kovalev.fsu@mail.ru

The problem of software implementation multiversioned views runtime data processing algorithms in information and control systems is discussed. The possibility of using existing multiversioned views models to improve the reliability of the software (SOFTWARE) for implementing the runtime of algorithms for data processing in control systems is shown.

The determining element of multiversioned views of the system unit is a decision about the correctness or incorrectness of States multiverse. This unit shares the results (outputs) of multiple software versions on the "correct" and "incorrect". There are several methods of such separation. The most common of them are based on the classification outputs. The most promising of these techniques are voted by an absolute majority vote agreed upon by the majority, and fuzzy voting agreed upon by the majority. In the framework of suggested improvements to these techniques, namely, weighted voting agreed upon by the majority and fuzzy weighted voting agreed upon by the majority. Methods of this group are based on the comparison of the outputs of multiverse and placing identical of them in the same classes (subsets). However, in cases where calculations are not performed with integers, the determination of the identity of outputs is difficult. To resolve this problem, the notion of value equality: we will say that two numbers are equal if they differ by less than some acceptable deviation. These ratios do not require the properties of transitivity. In other words, if it is known that $|a\ b| < \varepsilon$ *and* $|b\text{-}c| < \varepsilon$*, then it does not follow that* $|a\text{-}c| < \varepsilon$*, where a, b and c is some number. A potential problem of the methods of this group is possible misclassification of outputs and, as a consequence of a wrong decision that, ultimately, can lead to failure of the control system.*

Thus, the problem of the software implementation of SMVI algorithms of information processing in control systems is the development of software tools to unify the application of multiversioned views of the approach to different software systems. This approach, unlike the present, allows you to execute the modules are not only using the methodology multiversioned views programming, but also using other common multiversioned views models: recovering blocks, agreed recovering blocks, t/(n-1)-version programmin multiversioned views and programming self-test.

Keywords: information processing, multiversioned views software, system management, fault tolerance.

Несмотря на то, что концепция отказоустойчивых вычислений существует уже достаточно давно [1–5], до недавних пор прерогатива отказоустойчивости оставалось за проектировщиками аппаратной части. Аппаратные структуры разрабатываются так, чтобы они могли сохранять работоспособность и соответствовать заданным показателям даже при возникновении отказов, как случайных, так и повторяющихся [6–8]. Однако отказы аппаратных компонентов являются только одним из источников ненадежности компьютерных систем, который существенно теряет свою значимость как способ повышения надежности по мере увеличения размеров и сложности программных компонентов. Использование аппаратных методов отказоустойчивости предполагает отсутствие ошибок в выбранной модели обеспечения надежности, а применяемые меры исключают только отказы компонент, а не ошибки самой модели. Основная причина этого заключается в том, что очень сложно определить причину возникновения отказа на уровне аппаратной части. Следовательно, очень сложно разработать эффективную модель. Программные отказы, напротив, всегда являются следствием несоответствия модели, и их частота напрямую зависит от логической сложности программной модели [9–11].

Модели повышения надежности главным образом связаны с введением избыточности в систему [12–14]. Поэтому до недавнего времени улучшение программной составляющей критичных по надежности систем управления существенно ограничивалось вычислительными ресурсами применяемых вычислительных устройств. Однако сейчас развитие техники достигло того уровня, когда вычислительная мощность выпускаемых микропроцессоров значительно превосходит требования отдельных задач. Поэтому одной из основных задач разработчиков программного обеспечения (ПО) становится создание таких алгоритмов разработки программных систем, которые обеспечивали бы устойчивость системы к программным и аппаратным сбоям [15]. В связи с этим возникает техническая проблема, заключающаяся в создании программных средств разработки отказоустойчивого программного обеспечения. Это требует развития процедур проектирования отказоустойчивого программного комплекса систем управления, что является актуальной научной проблемой.

Известны работы, в которых объектом исследования является программный комплекс критичной по надежности системы управления и обработки информации, а предметом исследования – среда исполнения программного комплекса этой системы [16–22]. Причем целью данных исследований является повышение отказоустойчивости среды исполнения программных комплексов систем управления за счет применения мультиверсионных моделей [23–27].

Рассмотрим проблемы применения методологии проектирования отказоустойчивого программного обеспечения для повышения качества работы алгоритмов в системах управления и обработки информации. Для решения основных задач программной реализации мультиверсионной среды исполнения алгоритмов управления и обработки информации требуется разработать программную систему, представляющую собой инструментарий, который позволяет с минимальными временными затратами повысить отказоустойчивость систем управления при решении каждой конкретной задачи. Данный программный инструментарий должен представлять собой универсальную среду мультиверсионного исполнения (СМВИ) модулей программного комплекса. Для это необходимо также оценить эффективность различных моделей проектирования отказоустойчивого программного обеспечения [28–33] и алгоритмов мультиверсионного голосования [27; 34–37].

Анализ требований к среде мультиверсионного исполнения. В рамках решаемых данным исследованием задач предъявляются следующие требования к разрабатываемой системе: простота, надежность, производительность, компактность и универсальность.

Простота включает в себя такие требования к СМВИ, как отсутствие избыточных компонентов, четкое разделение функций между модулями и легкость применения системы исполнения под конкретную задачу. Сложность и излишняя разветвленность структуры не только снижают производительность, но также увеличивают вероятность ошибки в самой среде. Удовлетворение этих требований также позволяет снизить материальные и временные затраты на внедрение системы в производственный процесс.

Производительность, в свою очередь, складывается из таких факторов, как отсутствие избыточных связей между структурными компонентами системы и минимальные потери процессорного времени, затрачиваемые на работу внутренних алгоритмов самой системы исполнения [3; 6; 35; 38–40]. Эта группа требований является одной из самых главных, потому что обеспечивает реальность времени исполнения программных модулей систем управления, основанных на принципах мультиверсионного программирования [33; 35]. Это очень важно, так как чем меньше времени потребляет СМВИ, тем больше его остается программным модулям, выполняющим математический расчет. А мультиверсии требуют больших процессорных мощностей как за счет собственной сложности, так и за счет количества мультиверсий. Чтобы удовлетворить данные требования, необходимо хорошо проработать программную модель системы и, по возможности, сократить число циклов и разветвлений.

Компактность *среды* означает низкое количество потребляемой памяти. В свою очередь, требуемый размер памяти складывается из памяти, занимаемой самим кодом среды мультиверсионного исполнения, и рабочей памяти (или памяти, требуемой для хранения внутренних данных). Это обеспечивается за счет оптимизации всех циклов и разветвлений исходного кода программы, осуществляемой как на этапе разработки программной модели системы, так и после этапа тестирования и отладки.

Под надежностью среды мультиверсионного исполнения программных модулей понимается способность своевременно выдавать корректный результат вычислений [41; 42]. Эта способность зависит от таких факторов, как:

– отсутствие каких-либо ошибок в программном коде самой среды;

– устойчивость СМВИ к ошибкам и отказам модулей;

– корректная обработка любых ошибочных ситуаций (например, когда невозможно принять решение, возникновение внутренней ошибки системы, нехватка ресурсов и т. д.).

Следствием недостаточной надежности могут быть ситуации, связанные с выдачей неверного результата решающим алгоритмом, что может привести к краху всей системы. Удовлетворение требованиям надежности осуществляется за счет избыточности кода для обработки всех ошибочных ситуаций (там, где они могут возникнуть) и повышения уровня инкапсуляции и скрытия данных.

Под универсальностью понимается способность создаваемой среды мультиверсионного исполнения программных модулей исполняться на различных компьютерных системах, а также способностью СМВИ работать с различными типами входных и выходных данных. Для этого требуется, в первую очередь, чтобы программа распространялась не в виде готовых исполняемых модулей, а в виде наборов исходных кодов. Это позволит компилировать программу под конкретную платформу в каждом конкретном случае. Второе требование заключается в том, что все обращения к системным функциям должны быть вынесены в отдельный модуль. Третьим требованием является отсутствие привязки функций СМВИ к внутренней структуре входных, выходных и промежуточных данных мультиверсий. Иными словами, среда должна воспринимать эти данные как буфер определенного размера, лишь передавая его от модуля к модулю, не извлекая из него никаких данных и не производя в нем никаких изменений. Эти требования позволяют значительно расширить область применимости создаваемой среды.

Резюмируя вышеизложенное, можно изобразить все требования так, как представлено на рис. 1.

Анализ возможностей применения мультиверсионной методологии к системе управления. Идея мультиверсионных методологий заключается во введении программной избыточности за счет использования нескольких различных программных модулей, эквивалентных по функциональному назначению, но не являющихся отказоустойчивыми. Во всех модулях системы управления (СУ) могут возникать ошибки, но для упрощения понимания рассмотрим случай только для модуля обработки данных, так как применение мультиверсионной методологии к модулям ввода и вывода будет происходить аналогично. Для организации работы обработки данных несколькими методами создадим внутри системы управления среду мультиверсионного исполнения программных модулей. Эта среда будет исполнять все варианты алгоритмов обработки данных на основе выбранной мультиверсионной методологии и выдавать согласованный результат. Тогда структура системы управления будет выглядеть так, как изображено на рис. 2.

Фактически СМВИ представляет собой мультиверсионную систему управления, к которой подключаются модули ввода/вывода для осуществления связи с объектом управления, т. е. является ядром СУ.

Выбор способа реализации программных модулей и их взаимодействия со средой исполнения. Итак, рассмотрев содержание структуры СУ, можно перейти к более сложному вопросу: определение типов связей между модулями системы. Для этого попробуем определить, в каких модулях может произойти непредвиденный отказ, последствия которого нужно предотвратить.

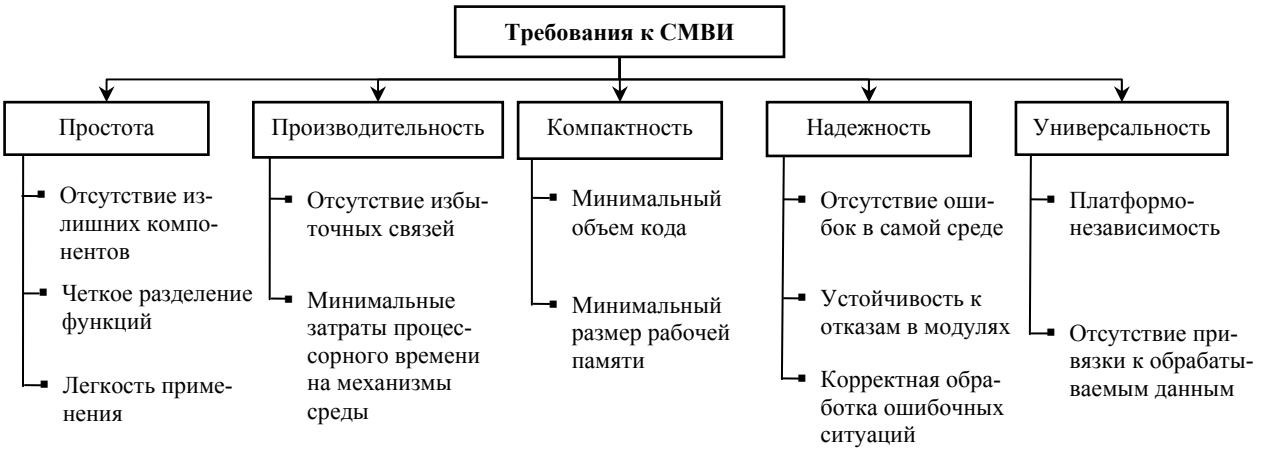

Рис. 1. Требования к разрабатываемому программному комплексу

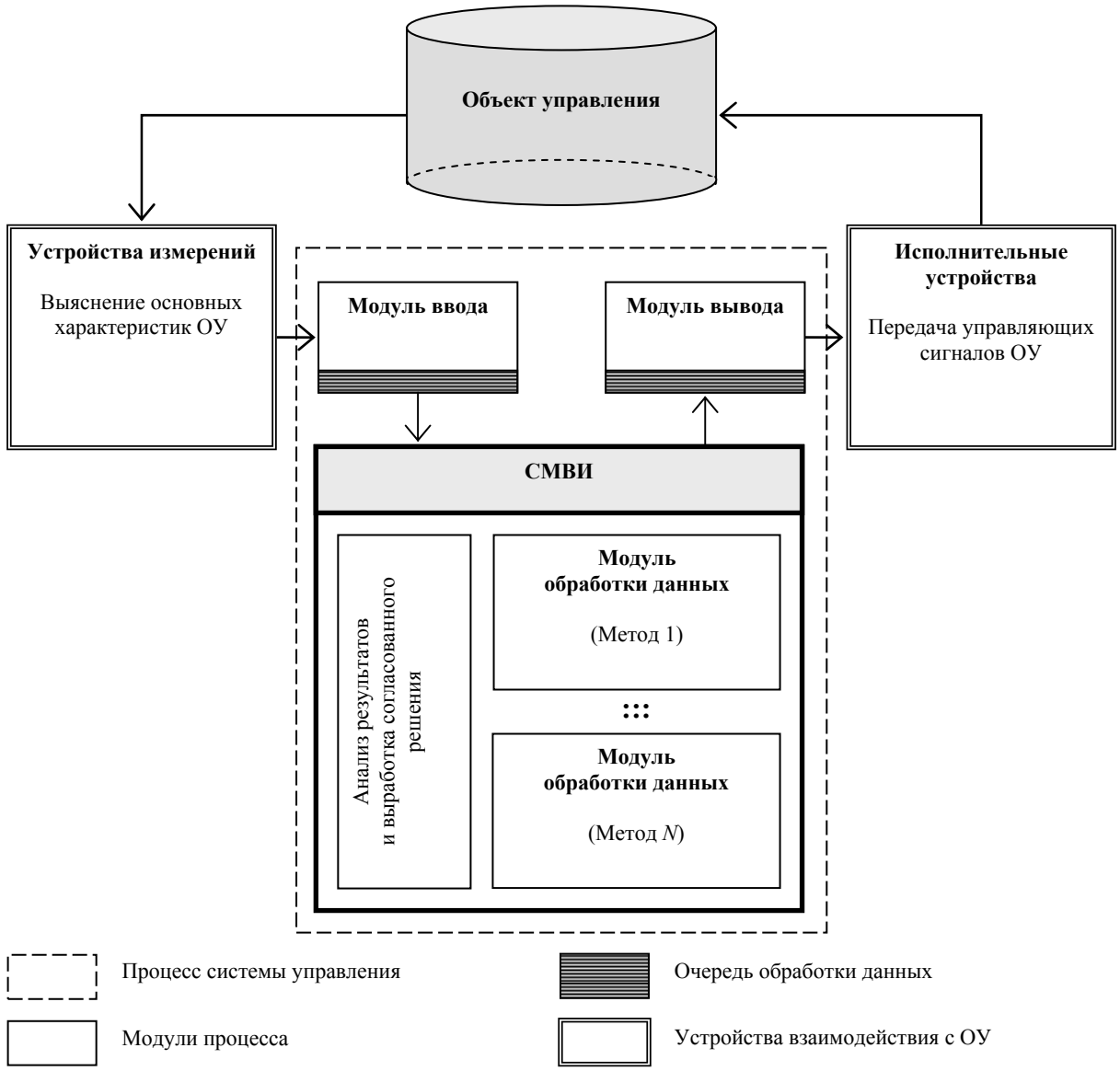

Рис. 2. Модель мультиверсионной системы управления с несколькими модулями обработки данных

Данные проекта загружаются в начале работы СУ, и (при необходимости) сохраняются при завершении – отказ при хорошей отладке произойти не может. Отказ исполнительного устройства (ИУ) или устройства измерений (УИ) в любом случае приведет к искажению данных и, как следствие, к отказу всей системы, поэтому обеспечить надежность универсальными программными средствами здесь невозможно. Модуль сравнения двух результатов содержит достаточно простой алгоритм, и простым тестированием этого модуля можно свести вероятность отказа в нем к нулю. Следовательно, связь этих модулей со средой исполнения можно реализовать через статическую компоновку или статические или динамические библиотеки. Таким образом, единственное «слабое звено» нашей системы – это мультиверсии [43], так как в них производятся сложные вычислительные операции. Здесь возможно несколько вариантов взаимодействия со средой (ниже они представлены в порядке возрастания потребности в памяти).

Статическая компоновка – из исходных кодов либо с использованием статических библиотек. Проект программной системы выглядит как набор исходных текстов (либо объектных кодов) СМВИ, в который требуется поместить под строго заданными именами исходные тексты (либо объектные коды) мультиверсий, ИУ, УИ, а также модуль сравнения двух результатов. После этого вызывается *makefile* – файл сборки проекта (в данном случае, хранящий данные проекта), и получается исполняемый модуль, готовый к исполнению.

Преимущества статической компоновки следующие.

1. Простые механизмы взаимодействия между модулями. Обращение к функции модуля является простым вызовом функции. Это позволяет снизить объем кода, а следовательно, и количество возмож-

ных ошибок. К тому же это самый быстрый из всех вариантов взаимодействия.

2. Эта модель обладает самой большой производительностью и самой низкой потребностью в памяти.

3. Данная модель требует от ОС (либо эмулирующего ее модуля) только наличия функций поддержки многопотоковости и объектов синхронизации.

К недостаткам отнесем следующее:

1. Общий процесс для всех модулей. Одно необработанное исключение в любом модуле приведет к отказу процесса, а следовательно, и всей системы управления.

2. Общее адресное пространство у версий и среды исполнения. При «зависании» одной из версий, СМВИ вынуждена будет ее перезапустить, т. е. удалить старый поток и создать его заново. Из-за общего адресного пространства операционная система (ОС) заблокирует весь процесс на время удаления потока (что может продлиться от миллисекунд до нескольких секунд), что может оказаться катастрофичным для объекта управления.

3. Накапливаемые утечки памяти. При перезапуске версии неизбежно будут возникать утечки памяти, которые могут привести к неработоспособности (либо отказу) как среды исполнения, так и перезапущенных версий. Бороться с ними можно, либо создав в СМВИ общие механизмы выделения и освобождения памяти и обязать использовать их мультиверсии (что несколько снизит производительность), либо использовать статическую память (что не всегда возможно).

4. Для внесения изменений в любую часть системы необходимо перекомпилировать всю систему и заново производить сборку проекта.

При компоновке с использованием динамических библиотек среда мультиверсионного исполнения представлена исполняемым модулем. Файлы с ИУ, УИ, версиями, модулем сравнения двух результатов и файл проекта выполнены в виде динамических библиотек со строго заданными в файле проекта именами. Для запуска такого проекта достаточно поместить все файлы в одну папку и запустить исполняемый модуль.

Преимущества при компоновке с использованием динамических библиотек:

– простые механизмы взаимодействия между модулями;

– данная модель не требует от ОС (либо эмулирующего ее модуля) наличия функций поддержки многопотоковости и объектов синхронизации.

Недостатки подхода:

– общий процесс для всех модулей;

– общее адресное пространство у версий и среды исполнения;

– накапливаемые утечки памяти.

В случае, когда компоненты выполнены в отдельных исполняемых модулях, для запуска проекта необходимо запустить все эти модули. Делать это вручную нецелесообразно, так как это ставит производительность среды в зависимость от оператора. Поэтому следует создать специальные программы-агенты, следящие за запуском всех модулей и перезапускающие их при необходимости. При использовании такой модели для исполнения всех модулей на одном компьютере агенты могут быть встроены в среду исполнения через дополнительный код в файле проекта. Но при сетевом распределении модулей встроить агентов в среду будет сложно, потому что возникнут трудности с отслеживанием состояния процесса на удаленной машине и его удаленного перезапуска.

Данную модель можно разделить на две, отличающиеся видом взаимодействия между исполняемыми модулями. Взаимодействие через общие участки памяти (например, файлы) обладает рядом преимуществ:

– нет утечек памяти при рестарте потоков;

– утечки памяти некатастрофичны;

– необработанное исключение приведет только к отказу одного модуля и, следовательно, безопасно для системы в целом.

Недостатки, характерные для данной модели, следующие:

1. Усложнение модулей с целью обеспечения синхронизации доступа к общим участкам памяти и, как следствие, возможное снижение надежности. К тому же могут возникнуть ситуации, способные привести к отказу системы. Например, отказ одного из модулей в момент передачи данных: помещенные в память данные некорректны из-за удаления процесса с модулем, последний уже никогда не сможет сообщить о завершении операции передачи данных среде, а среда не сможет определить, что ОС уничтожила модуль из-за отказа, полагая, что тот производит передачу данных. Именно такие ситуации невозможно предусмотреть, а по статистике более 90 % сбоев и отказов программ связано с некорректной работой памяти.

2. Необходимо написание специальных программ-агентов для рестарта модулей.

3. Данная модель компоновки модулей применима, только если ОС (либо эмулирующий ее модуль) поддерживает динамическое подключение библиотек.

Когда компоненты выполнены в отдельных исполняемых модулях и взаимодействие организовано через сетевой протокол, указанный подход также обладает рядом преимуществ:

– нет утечек памяти при рестарте потоков;

– утечки памяти некатастрофичны;

– необработанное исключение приведет только к отказу одного модуля и безопасно для системы в целом;

– возможность распределения вычислений по сети.

Однако это не исключает запуск и выполнение всех модулей на одной ЭВМ: современные сетевые протоколы позволяют производить соединение с самим собой, что нередко используется при разработке различного ПО.

К недостаткам можно отнести:

– необходимость написания специальных программ-агентов для рестарта модулей;

– возможность применения данной модели компоновки модулей только для ОС (либо эмулирующих ее модулей), поддерживающих работу с сетевыми протоколами.

В ряде работ [22–26; 44–46] были реализованы все рассмотренные варианты взаимодействия модулей со средой исполнения. Важно, что при сборке своего собственного проекта можно указать любой из них через директивы компилятора.

Конструирование программной модели. На основе представленной выше функциональной структуры системы мультиверсионного исполнения программных модулей рассмотрим ее программную модель [32]. Для этого распределим функциональные блоки по C++ классам. Тогда программная модель будет выглядеть так, как показано в таблице. Данная модель классов позволяет программно реализовать все рассмотренные ранее модели взаимодействия программных модулей со средой исполнения. При статической компоновке объекты CAlgModule, CInput, COutput, CComparator хранятся в виде исходных кодов либо в статических библиотеках, встраиваемых в исполняемый модуль на этапе компиляции.

При компоновке с использованием динамических библиотек CProject загружает внешние библиотеки с требуемыми объектами. Возможен вариант, когда некоторые объекты (например, CInput и COutput) вставляются в проект статически, а остальные – динамически. Для этого необходимо в зависимости от конкретной задачи указать эти сведения в конструкторе объекта CProject. При этом компоненты выполнены в виде отдельных исполняемых модулей. И в этом случае в проект встраиваются вместо самих объектов модули взаимодействия с ними, а сами объекты через программы-агенты исполняются в виде отдельного процесса. Их взаимодействие реализуется посредством сетевого протокола либо общих участков памяти.

Реализация общих алгоритмов функционирования среды исполнения. При реализации среды мультиверсионного исполнения существует два момента, которым стоит уделить особое внимание: общий алгоритм функционирования СМВИ и алгоритм синхронизации потоков (одновременного запуска всех мультиверсий). От первого зависит внутренняя структура методов и, как следствие, показатели надежности и производительности. Второй является решающим в определении устойчивости поведения системы при критических нагрузках процессора.

При разработке общего алгоритма функционирования следует особое внимание уделить корректной обработке потенциальных ошибок – выделить этапы, которые могут привести к отказу всей системы. Каждый блок представляет собой комплекс мер, реализуемых различными модулями системы.

Алгоритм синхронизации потоков системы составлен на основе теории разработки мультизадачных систем [8; 47]. Он использует автоматически сбрасывающиеся сигнальные объекты и функции ожидания сигнала, предоставляемые всеми современными операционными системами.

Разработанные алгоритмы приведены на рис. 3 и 4.

Структура классов, образующих программную модель СМВИ

Класс	Функции
CProject	Класс проекта. Этот класс переопределяется при конструировании системы для задания конкретных функций и характеристик системы: – объект ввода данных; – размер структуры входных данных; – объект вывода данных; – размер структуры выходных данных; – количество алгоритмических модулей; – объекты алгоритмических модулей; – объект сравнения двух результатов; – размер структуры состояния одного модуля. Все остальные классы СМВИ на этапе своей инициализации используют эту информацию
CNVPEE	Основной объект, он конструирует всю систему и осуществляет связь между объектами CProject, CStatistics и CManager
CStatistics	Этот объект хранит статистику процесса выполнения расчета, рассчитывает и выдает по требованию других классов вероятность ошибочного результата конкретной версии
CManager	Играет роль диспетчера всей системы: подготавливает к работе, согласует функционирование всех остальных модулей
CInput	Отвечает за ввод данных, т. е. считывает показания устройств измерения
CLauncher	Осуществляет пошаговое выполнение расчета версий, помещение их состояний в статистику и корректирует неверные состояния версий по результатам блока решения
CAlgModule	Алгоритмический модуль (АМ). Исполняет роль версии расчета
CDecision	Блок решения. На данном этапе разработки решение принимается простым голосованием без памяти: самая большая группа равных состояний – истинна, а все остальные – ошибочны, независимо от того, как часто каждая версия возвращает ошибочный результат
CComparator	Сравнение двух состояний
COutput	Отвечает за вывод данных, т. е. за передачу результатов расчета исполнительным механизмам

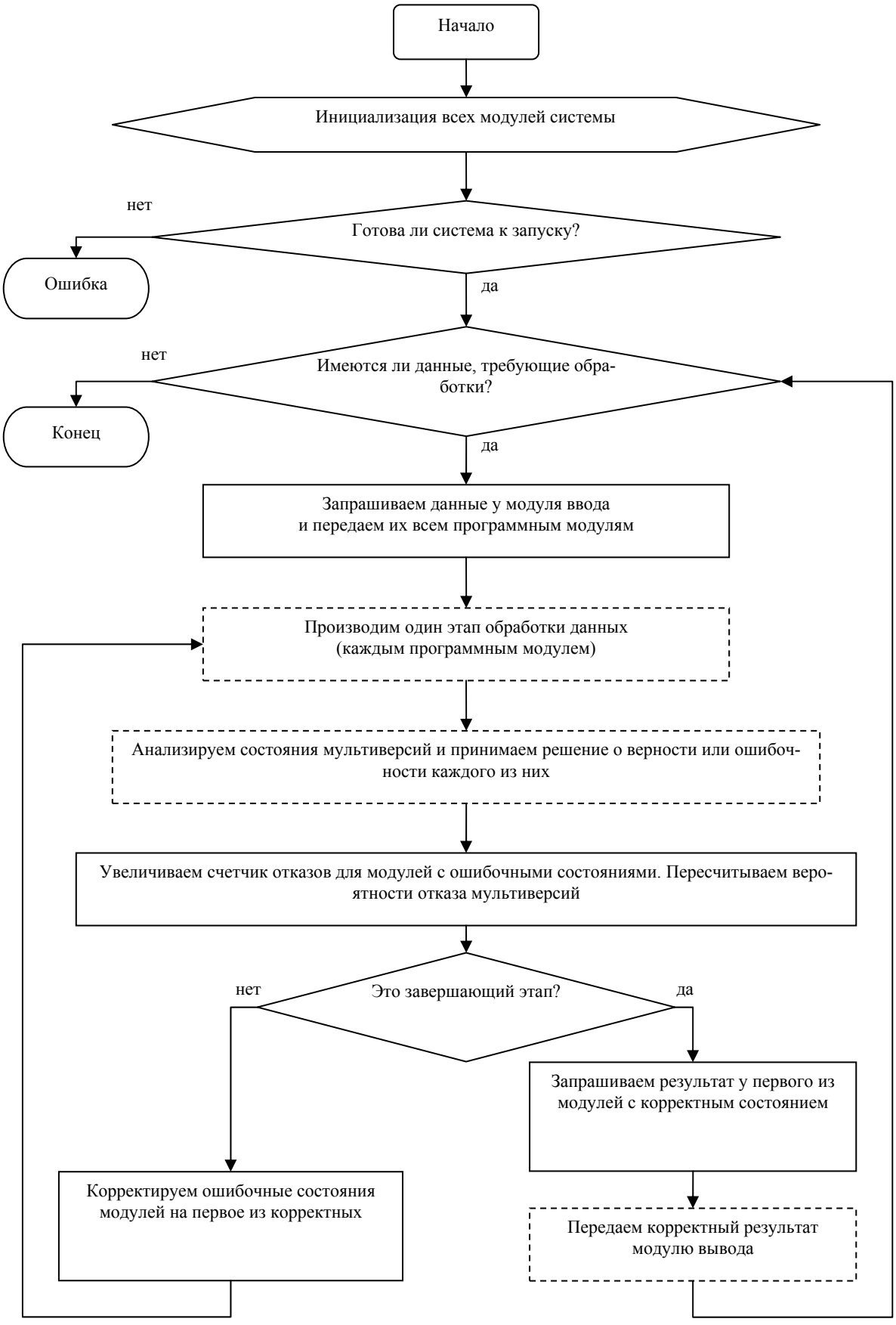

Рис. 3. Общий алгоритм функционирования СМВИ
(у блоков, выделенных пунктиром, имеется обработчик ошибочных ситуаций)

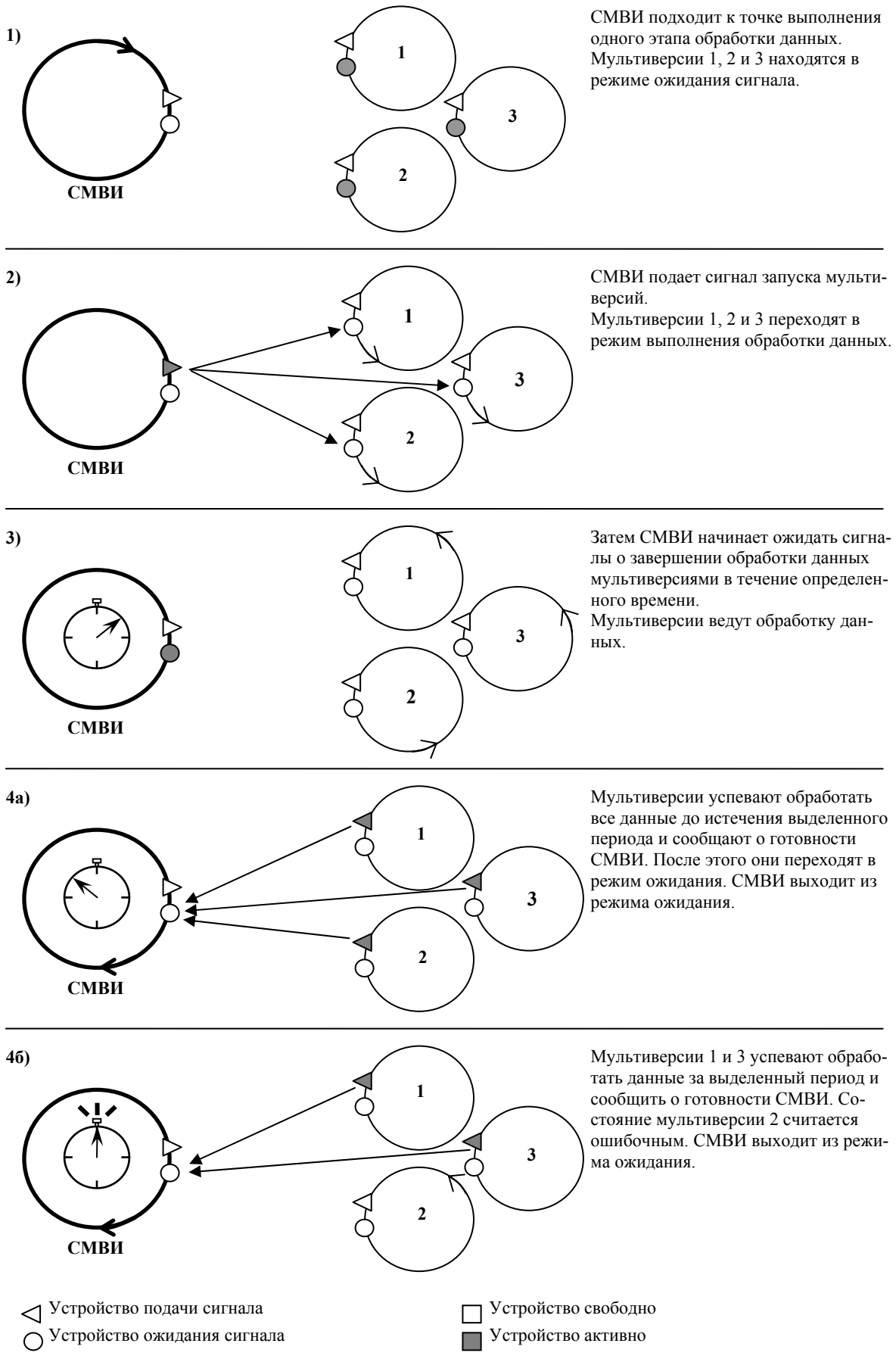

Рис. 4. Алгоритм выполнения одного этапа обработки данных

Реализация алгоритмов выявления отказов программных модулей и принятие решения. Во время исполнения программного обеспечения возможны два типа отказов. *Первый тип отказа* связан с нарушением нормального функционирования программы и возбуждением исключения. Это может произойти в случае попыток обращения по неверному адресу или деления на нуль. За перехват попыток обращения по неверному адресу и деления на нуль отвечает центральный процессор, возбуждающий исключения в ответ на эти ошибки. Исключение, возбужденное процессором, называется *аппаратным*. Но возбуждать исключения могут и операционная система, и прикладные программы. Такие исключения называются *программными*. Необработанное исключение в любом случае ведет к уничтожению процесса операционной системой. *Второй тип отказа* связан с зацикливанием программы. В этом случае программа на протяжении длительного времени не возвращает результат. В системах управления время является критическим показателем, поэтому необходимо ввести в среду исполнения такой параметр, как время отклика. Если программный модуль в течение длительного времени, большего чем заданное время отклика, не возвращает результат, то среда считает этот модуль отказавшим. Выявление отказов именно таким способом позволит нам отбросить часть неверных результатов и не использовать их в решающем алгоритме.

Определяющим звеном мультиверсионной системы является блок принятия решения о корректности или ошибочности состояний мультиверсий. Этот блок разделяет результаты (выходы) многочисленных программных версий на *корректные* и *ошибочные* [27]. Существует несколько методик подобного разделения [21–28]. Самые общие из них основываются на классификации выходов. Наиболее перспективными из этих методик являются *голосование абсолютным большинством*, *голосование согласованным большинством*, а также *нечеткое голосование согласованным большинством*. В рамках работы [27] предложены улучшения этих методик, а именно *взвешенное голосование согласованным большинством* и *нечеткое взвешенное голосование согласованным большинством*. Методики этой группы основаны на сравнении выходов мультиверсий и размещении идентичных из них в одинаковые классы (подмножества). Однако в случаях, когда расчеты выполняются не с целыми числами, определение идентичности выходов является затруднительным. Для разрешения этой проблемы введено понятие *соотношения равенства*: мы будем говорить, что два числа равны, если они отличаются меньше, чем на некоторое допустимое отклонение. Эти соотношения не требуют выполнения свойства транзитивности. Иными словами, если известно, что $|a - b| < \varepsilon$ и $|b - c| < \varepsilon$, то отсюда не следует что $|a - c| < \varepsilon$, где a, b и c – некоторые числа. Потенциальной проблемой методик данной группы является возможная неверная классификация выходов и, как следствие, неверное принятие решения, что, в конечном счете, способно привести к отказу системы управления.

Вторая группа методик принимает решение без классификации выходов. В эту группу входят: *медианное голосование* и *взвешенное медианное голосование*. Они выбирают среднее значение из всех выходов как корректное [27].

Заключение. Таким образом, решение проблемы программной реализации СМВИ алгоритмов обработки информации в системах управления заключается в разработке программного инструментария, позволяющего унифицировать применение мультиверсионного подхода к различным программным комплексам. Данный подход, в отличие от существующих в настоящее время, позволяет исполнять модули не только с помощью методологии мультиверсионного программирования, но также и с использованием других распространенных мультиверсионных моделей: восстанавливающихся блоков, согласованных восстанавливающихся блоков, $t/(n-1)$-версионного программирования [31] и мультиверсионного программирования с самопроверкой.

Библиографические ссылки

1. Avizienis A. Fault tolerance and fault intolerance: complementary approaches to reliable computing // In Proc. 1975 Intern. Conf. on Reliable Software. 1975. С. 458–464.

2. Avizienis A., Lyu M. R. The methodology of N-version programming. In Software fault-tolerance. Wiley, 1995. С. 23–47.

3. DEDIX 87 – A supervisory system for design diversity experiments at UCLA / A. Avizienis [et al.] // In Digest of 18th FTCS. Токио, 1988. С. 129–168.

4. Lyu M. R., He Y. Improving the N-Version Programming Process Through the Evolution of a Design Paradigm // IEEE COMSPAC, 1993.

5. Lyu M. R., Chen J. H., Avizienis A. Software diversity metrics and measurements // In Proc. IEEE COMPSAC 1992. Chicago, 1992. С. 69–78.

6. Fetzer C., Högstedt K., Felber P. Automatic Detection and Masking of Non-Atomic Exception Handling. 2003.

7. Mitchell G., Brown S. An approach for network communications systems Recovery. Department of Computer Science, National University of Ireland, 2000.

8. Romanovsky A. Diversely Designed Classes for Use by Multiple Tasks. University of Newcastle upon Tyne, 2000.

9. Ковалев И. В., Юнусов Р. В. Мультиверсионный метод повышения программной надежности информационно-телекоммуникационных технологий в корпоративных структурах // Дистанционное и виртуальное обучение. 2003. № 2. С. 50–55.

10. Kovalev I., Popov A., Shipovalov Ju. Optimization models for reliability of telecommunication software systems // Advances in Modeling and Analysis B: Signals, Information, Data, Patterns. 2000. Т. 43, № 3–4. С. 41–46.

11. Ковалев И. В., Завьялова О. И., Сисько В. Е., Царев М. Ю. Многоатрибутивное формирование оптимальных по составу высоконадежных сложных систем / Красноярский государственный аграрный университет. Красноярск, 2009.

12. Avizienis A., Chen L. On the implementation of N-version programming for software fault-tolerance during execution // In Proc. IEEE COMPSAC 77. 1977. С. 149–155.

13. Ковалев И. В., Завьялова О. И., Лайков А. Н. Формирование избыточного программного обеспечения отказоустойчивых систем управления // Известия высших учебных заведений. Приборостроение. 2008. Т. 51, № 10. С. 30–34.

14. Ковалев И. В., Семенько Т. И., Царев Р. Ю. Методология оценки и повышения надежности программно-информационных технологий и структур : монография / Федер. агентство по образованию ; Краснояр. гос. техн. ун-т. Красноярк, 2005.

15. Laprie J. C. Architectural Issues in Software Fault Tolerance, in Software Fault Tolerance / M. R. Lyu, editor. Wiley, 1995. Pp. 47–80.

16. Kovalev I. V., Younoussov R. V. Fault-tolerant software architecture creation model based on reliability evaluation // Advanced in Modeling & Analysis. Journal of AMSE Periodicals. 2002. Vol. 48, № 3–4. Pp. 31–43.

17. Ковалев И. В., Новой А. В. Анализ надежности программной архитектуры с учетом одновременного отказа компонентов // Приборы. 2009. № 7. С. 26–30.

18. Elmendorf W. Fault-tolerant programming // In Digest of 2-nd FTCS. Newton, 1972. С. 79–83.

19. Peter J. Denning, Fault Tolerant Operating Systems // ACM Computing Surveys. 1976. Vol. 8, No. 4. Pp. 359–389.

20. Steen M., Hauck F. J., Tanenbaum A. S. A Scalable Location Service for Distributed Objects / Vrije Universiteit. Amsterdam, 1996.

21. Daniels F. The Reliable Hybrid Pattern: A Generalized Software Fault Tolerant Design Pattern. Department of Electrical & Computer Engineering, North Carolina State University, 1999.

22. Kovalev I. V., Engel E. A., Tsarev R. Ju. Programmatic support of the analysis of cluster structures of failure-resistant information systems // Automatic Documentation and Mathematical Linguistics. 2007. Т. 41, № 3. С. 89.

23. Зеленков П. В., Ковалев И. В., Брезицкая В. В. Инструментальные средства формирования мультиверсионной архитектуры отказоустойчивых программных систем / М-во сельского хоз-ва Российской Федерации ; Краснояр. гос. аграрный ун-т. Красноярск, 2011.

24. Ковалев И. В. Имитационная система для среды мультиверсионного исполнения программных модулей (программная система «ИС-СМВИ v1.0») // Компьютерные учебные программы и инновации. 2007. № 2.

25. Котенок А. В. Программная система «СМВИ v1.0» (Среда мультиверсионного исполнения программных модулей). Номер гос. регистрации 50200401366 от 25.11.2004. М. : ВНТИЦ, 2004.

26. Ковалев И. В., Котенок А. В. Имитационная система для среды мультиверсионного исполнения программных модулей (программная система «ИС-СМВИ v1.0»). Номер гос. регистрации 50200501597 от 24.11.2005. М. : ВНТИЦ, 2005.

27. Ковалев И. В., Котенок А. В. К проблеме выбора алгоритма принятия решения в мультиверсионных системах // Информационные технологии. 2006. № 9. С. 39–44.

28. Randell B., Xu J. The Evolution of the Recovery Block Concept. University of Newcastle upon Tyne, 1995.

29. Anderson T., Lea P. A. Fault Tolerance: Principles and Practice. Practice Hall, 1981.

30. Randell B. System structure for software fault tolerance // IEEE Trans Software Engineering. Т. SE-1. 1975. С. 220–231.

31. Xu J., Randell B. The t/(n–1)-VP Approach to Fault-Tolerant Software. University of Newcastle upon Tyne, 1998.

32. Xu J., Randell B., Zorzo A. F. Implementing Software-Fault Tolerance in C++ and Open C++: An Object-Oriented and Reflective Approach. Department of Computing Science, University of Newcastle upon Tyne, 2000.

33. Bondavalli A., Stankovic J., Strigini L. Adaptable Fault Tolerance For Real-Time Systems. Pisa : CNUCE-CNR, 1995.

34. Антамошкин А. Н., Ковалев И. В., Царев Р. Ю. Математическое и программное обеспечение отказоустойчивых систем управления и обработки информации / Мин-во сельского хоз-ва Российской Федерации ; Краснояр. гос. аграрный ун-т. Красноярск, 2011.

35. Ковалев И. В., Ковалев П. В., Скориков В. С., Гриценко С. Н. Оценка времени выполнения мультиверсионных программ на кластере с последовательной и параллельной архитектурой обмена данными // Вестник СибГАУ. 2009. № 2 (23). С. 79–83.

36. Ковалев И. В., Ступина А. А., Царев Р. Ю., Волков В. А. Применение COM-технологии для реализации мультиверсионного программного обеспечения систем управления и обработки информации // Приборы и системы. Управление, контроль, диагностика. 2007. № 3. С. 18–22.

37. Kovalev I. V., Slobodin M. Ju., Tsarev R. Ju. Multi-version design of fault-tolerant software in control systems // Problems of mechanical engineering and automation. 2006. № 6. С. 61–69.

38. Минимизация межмодульного интерфейса для обеспечения надежности мультиверсионного программного комплекса / И. В. Ковалев [и др.] // Вестник СибГАУ. 2013. № 2 (48). С. 35–37.

39. Ковалев И. В., Антамошкин А. Н., Ерыгин В. Ю. Выбор структуры мультиверсионного программного обеспечения при нечетком бюджете и ограничениях на совместимость версий // Перспективы развития информационных технологий. 2012. № 7. С. 61–67.

40. Ковалев И. В., Царев Р. Ю., Капулин Д. В. Архитектурная надежность программного обеспечения информационно-управляющих систем / Мин-во сельского хоз-ва Российской Федерации ; Краснояр. гос. аграрный ун-т. Красноярск, 2011.

41. Ковалев И. В., Новой А. В., Штенцель А. В. Оценка надежности мультиверсионной программной архитектуры систем управления и обработки информации // Вестник СибГАУ. 2008. № 3. С. 50–52.

42. Ковалев И. В., Новой А. В. Расчет надежности отказоустойчивых архитектур программного обеспечения // Вестник СибГАУ. 2007. № 4. С. 14–17.

43. Ковалев И. В., Царев Р. Ю., Завьялова О. И. Анализ архитектурной надежности программного обеспечения информационно-управляющих систем // Приборы. 2010. № 11. С. 24–26.

44. Ковалев И. В., Нургалеева Ю. А., Ежеманская С. Н., Ерыгин В. Ю. Многоатрибутивное управление трудозатратами на разработку *n*-вариантных программных систем // Фундаментальные исследования. 2011. № 8–1. С. 124–127.

45. Ковалев И. В., Слободин М. Ю., Ступина А. А. Математическая постановка задачи проектирования *n*-версионных программных систем // Проблемы машиностроения и автоматизации. 2005. № 3. С. 16–23.

46. Kovalev I. V., Dgioeva N. N., Slobodin M. Ju. The mathematical system model for the problem of multiversion software design // Proceedings of Modelling and Simulation, MS'2004 AMSE : Intern. Conf. on Modelling and Simulation, MS'2004 / AMSE, French Research Council, CNRS, Rhone-Alpes Region, Hospitals of Lyon. Lyon-Villeurbanne, 2004.

47. Wattanapongsakorn N. Reliability Optimization for Software Systems with Multiple Applications. FastAbstract ISSRE and Chillarege Corp. Copyright, 2001.

References

1. Avizienis A. [Fault tolerance and fault intolerance: complementary approaches to reliable computing]. *In Proc. 1975 International Conference on Reliable Software.*1975, p. 458–464.

2. Avizienis A. The methodology of N-version programming. In Software fault-tolerance. Wiley, 1995, p. 23–47.

3. Avizienis A. DEDIX 87 – A supervisory system for design diversity experiments at UCLA. In Digest of 18th FTCS, Tokyo, Japan, June 1988, p. 129–168.

4. Lyu M. R. Improving the N-Version Programming Process Through the Evolution of a Design Paradigm. In Proc. IEEE COMSPAC, 1993.

5. Lyu M. Software diversity metrics and measurements. In Proc. IEEE COMPSAC. Chicago, Illinois, 1992, p. 69–78.

6. Fetzer C. Automatic Detection and Masking of Non-Atomic Exception Handling. 2003.

7. Mitchell G. An approach for network communications systems Recovery. Department of Computer Science, National University of Ireland, 2000.

8. Romanovsky A. Diversely Designed Classes for Use by Multiple Tasks. University of Newcastle upon Tyne, 2000.

9. Kovalev I. V., Junusov R. V. [Multiversioned views method for increasing software reliability information and telecommunication technologies in corporate structures]. *Distantsionnoe i virtual'noe obuchenie*. 2003, no. 2, p. 50–55 (In Russ.).

10. Kovalev I., Popov A., Shipovalov Ju. Optimization models for reliability of telecommunication software systems. Advances in Modeling and Analysis B: Signals, Information, Data, Patterns. 2000. vol. 43. no. 3–4, p. 41–46.

11. Kovalev I. V., Zav'jalova O. I., Sis'ko V. E., Carev M. Ju. *Mnogoatributivnoe formirovanie optimal'nykh po sostavu vysokonadezhnykh slozhnykh system* [Diversified formation of the optimal structure of highly reliable complex systems]. Ministry of agriculture of the Russian Federation, Krasnoyarsk state agrarian University, Krasnoyarsk, 2011.

12. Avizienis A. On the implementation of N-version programming for software fault-tolerance during execution. In Proc. IEEE COMPSAC 77, 1977, p. 149–155.

13. Kovalev I. V., Zav'jalova O. I., Lajkov A. N. [The formation of excessive software fault-tolerant control systems]. *Priborostroenie.* 2008, vol. 51, no. 10, p. 30–34 (In Russ.).

14. Kovalev I. V. *Metodologija otsenki i povyshenija nadezhnosti programmno-informatsionnykh tekhnologij i struktur* [The methodology for assessing and improving the reliability of software and information technologies and structures]. Krasnoyarsk, 2005.

15. Laprie J. C. Architectural Issues in Software Fault Tolerance, in Software Fault Tolerance. Wiley, 1995, p. 47–80.

16. Kovalev I. V. Fault-tolerant software architecture creation model based on reliability evaluation. Journal of AMSE Periodicals, 2002, vol. 48, no. 3–4, p. 31–43.

17. Kovalev I. V., Novoj A. V. [Reliability analysis of software architecture given the simultaneous failure of the components]. *Pribory*. 2009, no. 7, p. 26–30 (In Russ.).

18. Elmendorf W. Fault-tolerant programming. *In Digest of 2-nd FTCS*, Newton, June 1972, p. 79–83.

19. Peter J. Denning, Fault Tolerant Operating Systems. ACM Computing Surveys, December 1976, vol. 8, no. 4, p. 359–389.

20. Steen M. Scalable Location Service for Distributed Objects .Vrije Universiteit, Amsterdam, 1996.

21. Daniels F. The Reliable Hybrid Pattern: A Generalized Software Fault Tolerant Design Pattern. Department of Electrical & Computer Engineering, North Carolina State University, 1999.

22. Kovalev I. V., Engel E. A., Tsarev R. Ju. Programmatic support of the analysis of cluster structures of failure-resistant information systems. Automatic Documentation and Mathematical Linguistics. 2007, vol. 41, no. 3, p. 89.

23. Zelenkov P. V., Kovalev I. V., Brezickaja V. V. *Instrumental'nye sredstva formirovanija mul'tiversionnoj arkhitektury otkazoustojchivyh programmnykh sistem* [Tools of the formation multiversioned views of the architecture of fault-tolerant software systems]. Ministry of agriculture of the Russian Federation, Krasnoyarsk state agrarian University, Krasnoyarsk, 2011.

24. Kovalev I. V. [Simulation system for environment multiversioned views the execution of software modules (software system “IP-SMVI v1.0”)]. *Komp'juternye uchebnye programmy i innovatsii*. 2007, no. 2 (In Russ.).

25. Kotenok A. V. [Software system “SMVI v1.0” (Wednesday multiversioned views the execution of software modules)]. *VNTIC*, Moscow, 2004 (In Russ.).

26. Kovalev I. V., Kotenok A. V. *Imitatsionnaya sistema dlya sredy mul'tiversionnogo ispolneniya programmnykh modulei (programmnaya sistema“IS-SMVI*

v1.0"). [Simulation system for environment multiversioned views the execution of software modules (software system "IP-SMVI v1.0")]. No. 50200501597, VNTIC, Moscow, 2005 (In Russ.).

27. Kovalev I. V. [To the problem of algorithm selection decision-making systems multiversioned views]. *Informacionnye tehnologii.* 2006, no. 9, p. 39–44 (In Russ.).

28. Randell B. The Evolution of the Recovery Block Concept. University of Newcastle upon Tyne, England, 1995.

29. Anderson T. Fault Tolerance: Principles and Practice. Practice Hall, 1981.

30. Randell B. System structure for software fault tolerance. IEEE Trans Software Engineering, 1975, vol. 1, p. 220–231.

31. Xu J. The t/(n–1)-VP Approach to Fault-Tolerant Software. University of Newcastle upon Tyne, 1998.

32. Xu J. Implementing Software-Fault Tolerance in C++ and Open C++: An Object-Oriented and Reflective Approach. Department of Computing Science, University of Newcastle upon Tyne, 2000.

33. Bondavalli A. Adaptable Fault Tolerance For Real-Time Systems. CNUCE-CNR, Pisa, Italy, 1995.

34. Antamoshkin A. N., Kovalev I. V., Carev R. Ju. *Matematicheskoe i programmnoe obespechenie otkazoustojchivyh sistem upravlenija i obrabotki informatsii* [Mathematical and software fault-tolerant control systems and information processing]. Ministry of agriculture of the Russian Federation, Krasnoyarsk state agrarian University, Krasnoyarsk, 2011.

35. Kovalev I. V., Kovalev P. V., Skorikov V. S., Gricenko S. N. [The estimation of the execution time multiversioned views programs on a cluster with serial and parallel architecture for data exchange]. *Vestnik SibGAU.* 2009, no. 2 (23), p. 79–83 (In Russ.).

36. Kovalev I. V., Stupina A. A., Carev R. Ju., Volkov V. A. [The use of COM technology for the implementation of multi-version software of control systems and information processing]. *Upravlenie, kontrol', diagnostika.* 2007, no. 3, p. 18–22 (In Russ.).

37. Kovalev I. V., Slobodin M. Ju., Tsarev R. Ju. Multi-version design of fault-tolerant software in control systems. *Problems of mechanical engineering and automation.* 2006, no. 6, p. 61–69.

38. Kovalev I. V., Nurgaleeva Ju. A., Shahmatov A. V., Chekmarev S. A., Lukin F. A. [To minimize inter-module interface to ensure the reliability multiversioned views software]. *Vestnik SibGAU*. 2013, no. 2 (48), p. 35–37 (In Russ.).

39. Kovalev I. V., Antamoshkin A. N., Erygin V. Ju. [The choice of patterns multiversioned views of the software during the fuzzy budget and limitations for version compatibility]. *Perspektivy razvitija informacionnyh tehnologij*. 2012, no. 7, p. 61–67 (In Russ.).

40. Kovalev I. V., Carev R. Ju., Kapulin D. V. *Arhitekturnaja nadezhnost' programmnogo obespechenija informacionno-upravljajushhikh system* [Architectural software reliability information management systems]. Ministry of agriculture of the Russian Federation, Krasnoyarsk state agrarian University, Krasnoyarsk, 2011.

41. Kovalev I. V., Novoj A. V., Shtencel' A. V. [To assess the reliability multiversioned views of software architecture, control systems and information processing]. *Vestnik SibGAU*. 2008, no. 3 (20), p. 50–52 (In Russ.).

42. Kovalev I. V., Novoj A. V. [Calculation of reliability of fault-tolerant software architectures]. *Vestnik SibGAU*. 2007, no. 4 (17), p. 14–17 (In Russ.).

43. Kovalev I. V., Carev R. Ju., Zav'jalova O. I. [Analysis of architectural software reliability information management systems]. *Pribory*. 2010, no. 11, p. 24–26 (In Russ.).

44. Kovalev I. V., Nurgaleeva Ju. A., Ezhemanskaja S. N. [Diversified management work on the development of n-variant software systems]. *Fundamental'nye issledovanija*. 2011, no. 8, p. 124–127 (In Russ.).

45. Kovalev I. V., Slobodin M. Ju., Stupina A. A. [Mathematical formulation of the problem of designing n-version software systems]. *Problemy mashinostroenija i avtomatizatsii*. 2005, no. 3. p. 16–23 (In Russ.).

46. Kovalev I. V., Dgioeva N. N., Slobodin M. Ju. The mathematical system model for the problem of multiversion software design. Proceedings of Modelling and Simulation, MS'2004 AMSE International Conference on Modelling and Simulation, MS'2004. AMSE, French Research Council, CNRS, Rhone-Alpes Region, Hospitals of Lyon. Lyon-Villeurbanne, 2004.

47. Wattanapongsakorn N. Reliability Optimization for Software Systems with Multiple Applications. FastAbstract ISSRE and Chillarege Corp. Copyright 2001.