СПОСОБЫ РАЗРАБОТКИ РАСПРЕДЕЛЕННЫХ АВТОМАТИЗИРОВАННЫХ СИСТЕМ ДИСПЕТЧЕРСКОГО УПРАВЛЕНИЯ ПОВЫШЕННОЙ НАДЕЖНОСТИ

Белоконь С.А., Васильев В.В., Золотухин Ю.Н., Филиппов М.Н., Ян А.П.
Институт автоматики и электрометрии СО РАН
630090, г. Новосибирск, просп. Академика Коптюга, 1
zol@idisys.iae.nsk.su

Предложен ряд способов повышения надежности автоматизированных систем диспетчерского управления (АСДУ), в частности: специализированная архитектура на основе равноправных серверов, динамический программный интерфейс, организующий взаимодействие между модулями распределенной SCADA-системы, а также вариант трехзначной логики, существенно упрощающий запись логических выражений при обработке неполной и недостоверной информации. Разработан метод оценки надежности программного обеспечения по информации, сохраняемой системой управления версиями, учитывающий индивидуальную историю создания.

Ключевые слова: распределенные системы, повышение надежности, автоматизированные системы диспетчерского управления, динамический программный интерфейс, трехзначная логика, система управления версиями

Введение

Автоматизированные системы диспетчерского управления, предназначенные для использования на объектах повышенной опасности, таких как предприятия атомной и химической промышленности, транспортные комплексы, объекты военного назначения и т. п., занимают особое место, так как нарушения в их работе представляют прямую угрозу жизни и здоровью людей.

При разработке как аппаратных средств, так и программного обеспечения (ПО) подобных систем недостаточно лишь организации дополнительного резервирования составляющих, поскольку помимо высокой надежности при работе в штатном режиме весь программно-аппаратный комплекс должен обеспечивать предсказуемо-безопасное поведение в случае выхода из строя отдельных компонентов.

Ограничения, налагаемые конкретными объектами управления, а также перечень требований, регламентируемых государственными, отраслевыми и внутренними стандартами предприятий, не позволяют найти полностью готовое универсальное решение, но, тем не менее, позволяют предложить базовую архитектуру аппаратного комплекса и принципы организации программного обеспечения, достаточные для построения надежных и безопасных систем диспетчерского управления объектами повышенной опасности.

Необходимость в математическом обеспечении, учитывающем варианты развития событий, вызвана также и отличием АСДУ от полностью автоматических систем: диспетчерское управление предполагает первостепенное и активное участие человека, поэтому анализ отдаваемых им команд и блокирование ошибочных действий в режиме реального времени является необходимым условием обеспечения безопасности, позволяющим уменьшить влияние так называемого «человеческого фактора».

Еще на этапе проектирования подобное усложнение логики работы может значительно затруднить оценку характеристик создаваемой системы, основными из которых являются надежность, безопасность и живучесть [1]. И если в случае аппаратной части, составленной из стандартных компонентов, можно воспользоваться предоставляемыми производителями данными о среднем времени наработки на отказ, сроке службы и т. п., то для программной части использование статистического подхода, по крайней мере на первом этапе, неприменимо ввиду уни-

кальности системы управления и планируемого штучного использования. Таким образом существует необходимость разработки принципов организации ПО, позволяющих составлять сложные и логически непротиворечивые программные комплексы из поддающихся анализу относительно простых модулей.

На основе изложенного можно сделать вывод, что основными требованиями при разработке систем управления объектами повышенной опасности являются:

- обеспечение повышенной надежности программно-аппаратного комплекса и безопасного поведения при отказах;
- ограничение несанкционированного доступа к управлению и данным системы;
- снижение негативного влияния человеческого фактора.

Особенности архитектуры

В настоящее время стандартной схемой резервирования является установка двух (реже трех) серверов, которые в документации названы «основным» и «резервным», но фактически работающих как «ведущий» и «ведомый», поскольку в штатном режиме только основной сервер взаимодействует с контроллерами, отправляя команды, и одновременно обменивается данными с резервным сервером, постоянно обновляя его статус. Если основной сервер выходит из строя или обмен данными с ним прекращается, резервный сервер берет на себя его функции и начинает обрабатывать запросы клиентов [2].

Данная архитектура, несмотря на предлагаемое значительное упрощение разработки программного обеспечения и повышенное быстродействие (в основном, благодаря снижению нагрузки на контроллер и внутреннюю сеть), не лишена ряда существенных недостатков. Например, при выполнении экстренной команды оператора несколько секунд задержки, необходимые для переключения управления от вышедшего из строя основного сервера на резервный, могут привести к возникновению опасной или даже аварийной ситуации. Проблема таких больших задержек при использовании данной архитектуры в территориально распределенных системах скорее алгоритмическая: время принятия решения об отказе основного сервера (или канала связи) хотя и желательно минимизировать, но нельзя сделать слишком малым. Если это время будет соизмеримо с задержками доставки информации от клиента к серверу и подтверждения — в обратном направлении, то это приведет к частым ложным тревогам и неоправданным переходам с одного сервера (или канала связи) на другой. Возможность хотя и редких коллизий в канале связи и задержек программного обеспечения при получении запроса и формировании ответа, а также иногда возникающая необходимость нескольких повторных передач заставляет увеличивать время принятия решения до нескольких секунд [3].

Также в случае, если по какой-то причине теряется связь между серверами, они оба переходят в статус «ведущего», что, как правило, приводит к чрезвычайно опасной рассинхронизации управления, при которой контроллер будет получать двойные (реакция серверов на происходящие события), а иногда и противоречащие друг-другу команды.

Для исключения подобных ситуаций предложена архитектура АСДУ, существенным отличием которой от классической схемы с иерархией «основной-резервный» является параллельная рассылка копий команд управления по нескольким независимым каналам (рис. 1). Принимаемые команды анализируются асинхронно работающими равноправными серверами перед отправкой на низлежащий уровень (контроллер); последний, в свою очередь, выполняет первую полученную команду и запоминает ее уникальный идентификатор вместе с результатом исполнения, которые затем используются при формировании ответов на поступающие позже копии.

Таким образом, при выходе из строя одного из каналов доставки и обработки команд, включая компьютеры, маршрутизаторы, оптоволоконный кабель, сетевой модуль ПЛК и т. п., обеспечивается выполнение задания и получение ответа на отправленный запрос. При этом отсутствуют обычные в таких случаях (и зачастую существенные — до нескольких секунд) задержки на ожидание перед переключением на резервное оборудование и повторной посыл-

кой сообщения, что обеспечивает не только повышенную отказоустойчивость системы, но и быстродействие системы, необходимое при выполнении экстренных команд оператора.

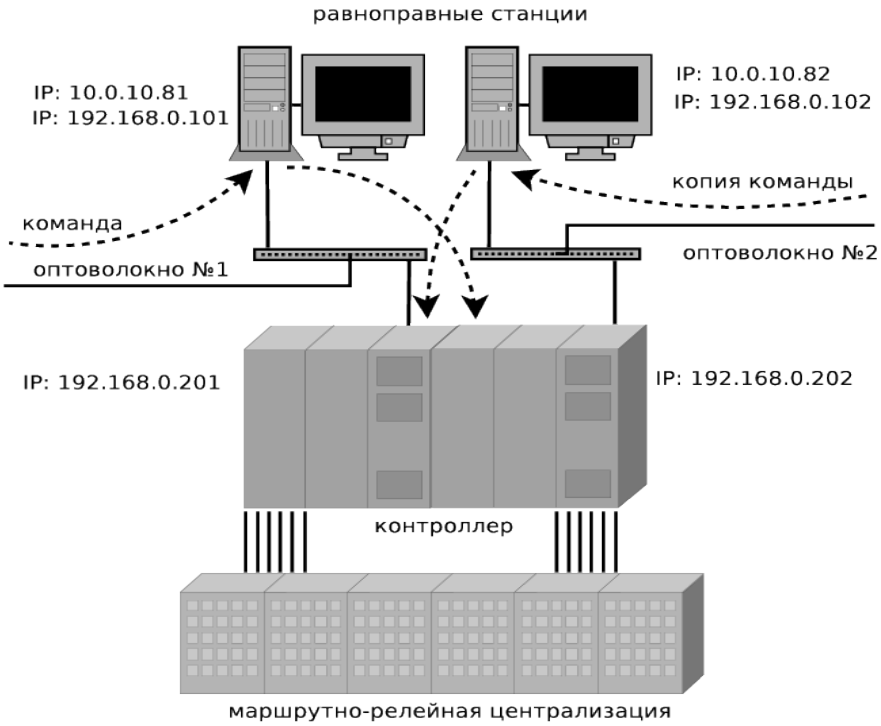

Рис. 1: Архитектура АСДУ на основе равноправных серверов

В качестве примера практического использования предложенной архитектуры представлен средний уровень АСДУ движением поездов метрополитена (подробное описание архитектуры приведено в работе [4]), одной из основных задач которого является предоставление информации поездному диспетчеру и передача команд на уровень объекта управления, причем отдаваемые команды анализируются программным обеспечением с целью выявления несогласованных и потенциально опасных действий, способных привести к аварийной ситуации. Для соблюдения данного требования каждое действие оператора анализируется независимо друг от друга тремя различными подсистемами.

Аппаратно первый и второй АРМы ДСЦП (автоматизированные рабочие места дежурных по станции) подключены к маршрутизаторам, расположенным в противоположных фойе станции метрополитена на значительном расстоянии друг от друга, причем каждый из маршрутизаторов соединен с обоими каналами оптоволоконного кольца, чем обеспечивается надежная передача данных и функционирование всей системы в случае отключения энергоснабжения, по жара или затопления в одном из серверных помещений, что повышает общий уровень катастрофоустойчивости.

На момент получения команды внутренние состояния модулей логики серверов в общем случае различны, поскольку, работая в асинхронном режиме, они опрашивают контроллер в различные моменты времени. Это значительно уменьшает вероятность одновременного отказа обоих узлов в случае ошибок, проявляющихся при определенных состояниях входов системы [5].

Главным достоинством предложенной архитектуры является обеспечение многоуровневого анализа безопасности действий оператора, повышенной живучести системы и восстановления рабочего состояния в случае выхода из строя ее составляющих. Дополнительное преимущество схемы с равноправными серверами состоит в возможности изменения конфигурации без перезапуска остальной части системы, например, подключение дополнительных резервирующих узлов и линий связи, выключение части оборудования на профилактическое обслужи-

вание, а также поэтапная замена программного и аппаратного обеспечения с предварительным тестированием в параллельном режиме.

Особенности программного обеспечения

В настоящее время подавляющее большинство SCADA-систем предназначено для использования исключительно в операционной системе Windows, а буквально единицы оставшихся либо не составляют им конкуренции в плане функциональности, либо разработаны для какой-либо конкретной операционной системы и не являются многоплатформенными. Это является серьезным препятствием при разработке распределенных систем, в которых к некоторым узлам предъявляются повышенные требования по отказоустойчивости (что требует использования хорошо зарекомендовавших себя решений на базе операционных систем QNX или GNU/Linux), а второстепенные узлы, реализующие только функции наблюдения, могут представлять собой стандартные персональные компьютеры с более привычной для пользователей и обслуживающего персонала операционной системой MS Windows.

Исследования надежности систем АСУ ТП показывают, что значительное количество критических ошибок, существенно снижающих уровень информационной безопасности управляющих комплексов, ежегодно выявляется как в операционной системе MS Windows, так и в коммерческих SCADA-системах [6].

Таким образом, не менее важным фактором при выборе программного обеспечения является возможность доступа к исходным текстам с целью непредвзятого анализа надежности, исследования недокументированных особенностей поведения и, при необходимости, независимого от разработчика и оперативного расширения функциональности.

Приведенные выше требования положены в основу спецификаций открытой многоплатформенной SCADA-системы. При ее разработке активно использован принцип модульности (построение из автономных элементов с простыми и согласованными структурными связями между ними) для обеспечения гибкости и расширяемости архитектуры, упрощения повторного использования кода, а также увеличения надежности программы.

Внутренняя архитектура SCADA-системы подчинена разработанной концепции динамического программного интерфейса, основной идеей которого является организация всех разделяемых между частями системы точек взаимодействия в виде единой динамически изменяющейся структуры. С точки зрения модуля регистрация (или публикация) собственного интерфейса выглядит как операция создания нескольких элементов в специализированной виртуальной файловой системе, поддерживающей функции работы с объектами, организованными в виде древовидной структуры с узлами-каталогами, а в качестве имени-идентификатора выступает строка с символом-разделителем между элементами – путь от корневого каталога до конкретного объекта [7].

В качестве иллюстрации на рис. 2 приведена типовая конфигурация ПО АРМ дежурного по станции [4], состоящая из шести одновременно работающих модулей, которые взаимодействуют по протоколу TCP/IP: модуль маршрутизатора (hub) предназначен для пересылки сообщений между компонентами системы; модуль пользовательского интерфейса (viewer) обеспечивает отображение графической информации на экране оператора и передачу вводимых команд модулю логики (logic) для дальнейшей обработки; модуль связи (modbus) пересылает проверенные модулем логики команды в контроллер, а также периодически опрашивает состояние переменных контроллера и генерирует сообщения об изменениях; модуль базы данных (shn writer) заносит в нее записи об изменении состояния системы; наконец, модуль dch server обеспечивает связь с верхним уровнем АСДУ.

Существенное сокращение времени разработки приложений достигнуто за счет поддержки открытых стандартов. Например, поскольку все конфигурационные файлы системы, включая описания отображаемых на экране объектов, основаны на формате XML, разработчику предоставляется возможность выбора не только наиболее удобного в каждом конкретном случае приложения-редактора, но и системы управления версиями для работы со всеми файлами программного продукта, включая графическую информацию.

Интегрированная поддержка сценариев позволяет реализовать концепцию выработки для конкретной задачи предметно-ориентированного языка (DSL — Domain-Specific Language), значительно сократить объем работы программиста, а также повысить надежность программного обеспечения благодаря корректной обработке интерпретатором типичных ошибок исполнения.

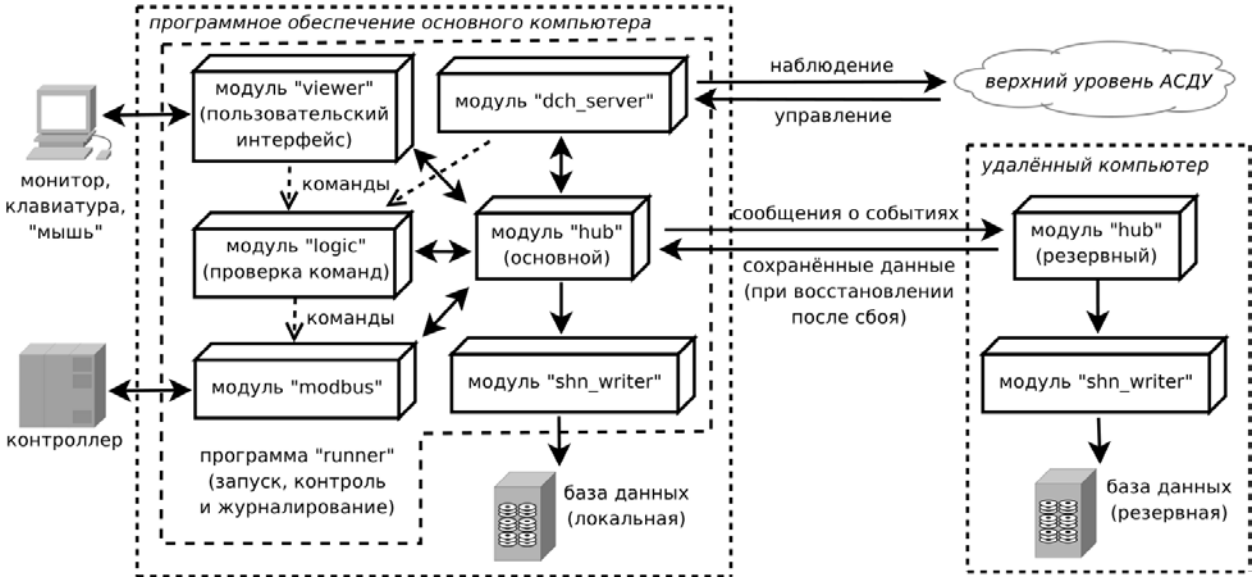

Рис. 2: Схема взаимодействия модулей ПО АРМ дежурного по станции

Предложенная архитектура позволяет разрабатывать программные комплексы со сложными логическими зависимостями между частями, составляя динамическую конфигурацию из подгружаемых модулей.

Метод оценки надежности программного обеспечения

Для оценки надежности программного обеспечения использован метод, позволяющий учесть особенности индивидуальной истории создания программы, заключающийся в выявлении зависимости количества значимых изменений кода от фактического времени разработки по предоставляемой журналом системы управления версиями информации и экстраполяции полученной функции с целью получения числовых оценок остаточного количества ошибок и скорости их обнаружения [8].

В качестве иллюстрации представлен пример практического применения предложенного метода для оценки характеристик программного обеспечения системы управления движением поездов станции «Заельцовская» Новосибирского метрополитена. При разработке программного обеспечения использована хорошо зарекомендовавшая себя открытая централизованная система управления версиями Subversion.

На рисунке 3 приведён построенный по записям журнала системы управления версиями график времени выпуска версии программы, начиная с момента получения предварительных спецификаций 28 сентября 2007 г. и заканчивая установкой программы на станции в режим опытной эксплуатации 21 мая 2008 г.; всего за этот период в исходный текст программы было внесено 407 изменений. В нижней части рисунка для удобства дальнейшего рассмотрения исключены наиболее заметные интервалы времени, в течение которых разработка программы была приостановлена по различным причинам, таким как участие сотрудников в других проектах и конференциях, праздничные дни, дни отпуска и т. п.

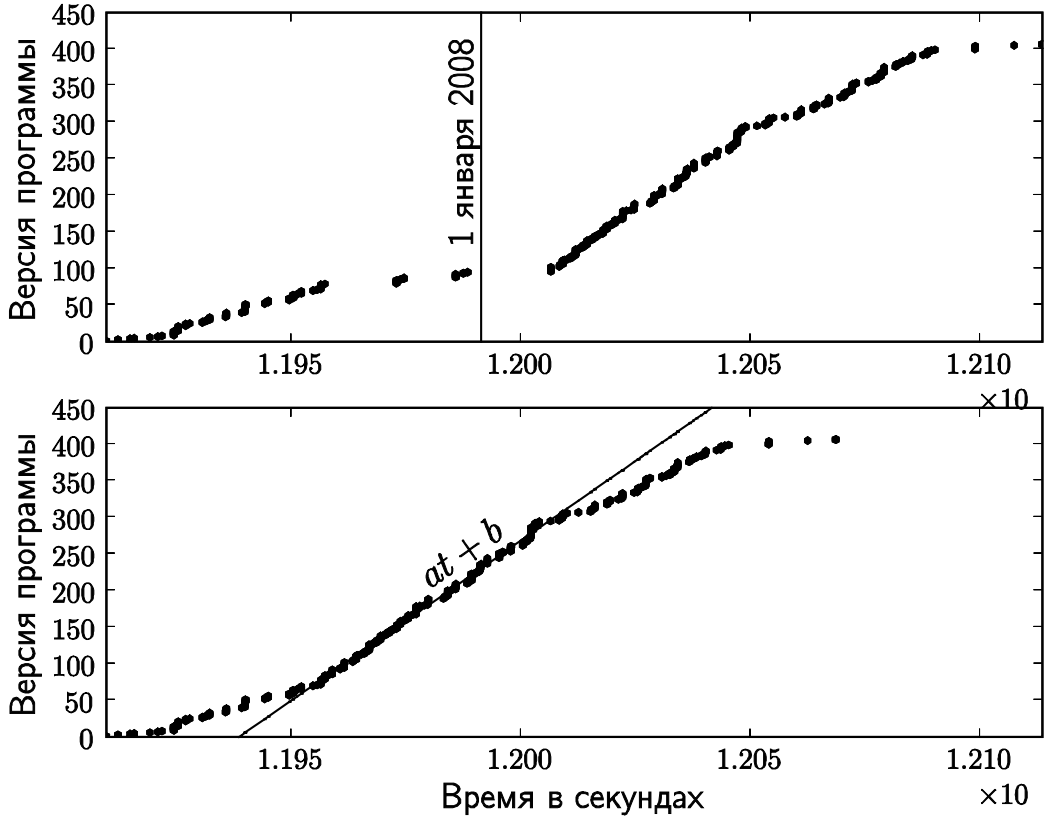

Рис. 3: Версия программы от времени разработки

При анализе данного графика целесообразно выделить три основные части: первая соответствует начальному периоду разработки, во время которого проводится уточнение спецификаций, планируется внутренняя архитектура программного обеспечения и создаются прототипы основных модулей. На данном этапе скорость внесения изменений относительно невелика, затем она постепенно увеличивается и достигает максимального значения к средней части графика.

В течение следующего периода, который соответствует времени активной разработки, кривая подчиняется простой линейной зависимости *(at + b)*. Объясняется это тем, что, несмотря на готовый план работы и четкие спецификации программных интерфейсов, скорость внесения изменений ограничена производительностью труда коллектива разработчиков.

Наконец, последняя часть графика вновь демонстрирует уменьшение скорости внесения изменений и отражает плавный переход от активной разработки к процессу тестирования, описываемому принципиально другой зависимостью (рис. 4), которая и представляет наибольший интерес с точки зрения оценки основных характеристик надежности поступающего в эксплуатацию программного обеспечения.

На представленном графике видно, что изменения обычно вносятся группами по 3–5 исправлений подряд. Отчасти это связано с особенностями работы в системе Subversion, но в основном объясняется тем, что требования об изменениях, предоставляются по окончании очередного периода тестирования в виде списка найденных ошибок без указания точного времени их обнаружения. Поэтому для построения аппроксимирующей функции использованы точки, описывающие конечные состояния периодов интенсивного исправления, между которыми прошло не менее одного рабочего дня.

В результате анализа широкого класса функций выяснено, что наблюдаемые данные на этом участке хорошо аппроксимируются зависимостью $N(1-e^{-(t/\lambda)^{\kappa}})$.

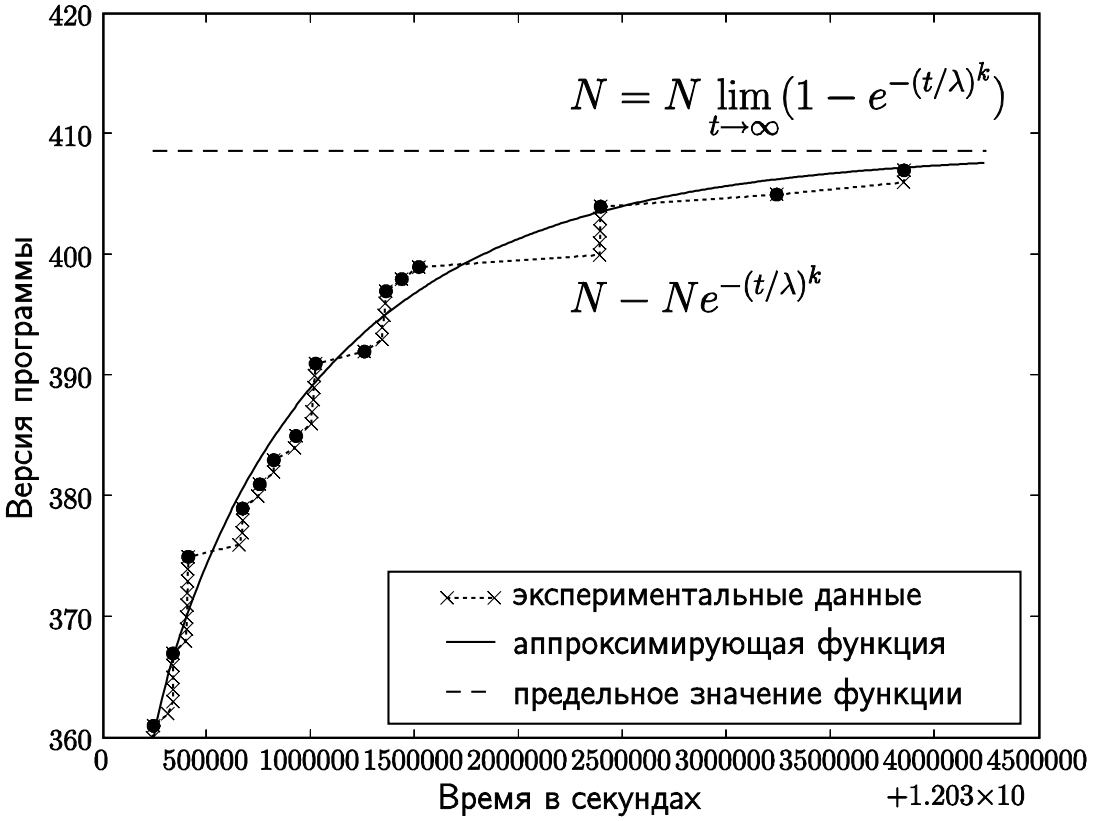

Рис. 4: Версия программы от времени тестирования

Исследование ряда других программ с сохранившимися журналами системы управления версиями подтвердило, что выявленная закономерность с хорошей степенью точности описывает период заключительного тестирования программного обеспечения, непосредственно предшествующий сдаче его в эксплуатацию.

При этом получаемые по методу наименьших квадратов коэффициенты (N, λ, k) позволяют оценить количество ошибок на момент начала тестирования, эффективность выбранных методик тестирования и практическую скорость выявления ошибок, а также качество конечного продукта по оценке количества ошибок, оставшихся невыявленными к моменту окончания работ, которая на основании существующего предела полученной функции имеет вид

(1) $n(t) = Ne^{-(t/\lambda)\kappa}$.

На основе приведенной зависимости получено выражение для ожидаемого количества ошибок $n(T_1, T_2) = N(e^{-(T_1/\lambda)\kappa} - e^{-(T_2/\lambda)\kappa})$, обнаруживаемых за период времени $T_1 - T_2$. В частности для рассматриваемого в качества примера программного обеспечения станции «Заельцовская» была получена оценка, что в течение запланированных 2 месяцев выявляются 98.5% оставшихся ошибок при условии сохранения интенсивности тестирования.

Для получения количественных характеристик надежности, в частности среднего времени наработки до отказа, использована модель нестационарного пуассоновского процесса:
введены вспомогательные функции интенсивности обнаружения ошибок $\mu(t)$ и количества остающихся не обнаруженными ошибок $n(t)$, тогда $dn(t) = -n(t)\ \mu(t)dt$, откуда $n(t) = N\ e^{-\int_0^t \mu(\tau)\,d\tau}$, где N – суммарное количество ошибок в программе на момент начала тестирования. После ряда преобразований полученная из зависимости (1) интенсивность обнаружения ошибок имеет вид $\mu(t) = k/t^{1-k}\ \lambda^k$.

Поскольку в анализируемом случае $k < 1$, интенсивность обнаружения ошибок $\mu(t)$ со временем уменьшается. Качественно это объясняется тем, что к моменту времени t наиболее очевидные и часто встречающиеся ошибки уже найдены и исправлены, а для обнаружения оставшихся требуется совпадение большого количества условий или возникновение редко проявляющихся факторов.

Для передаваемой в эксплуатацию программы можно оценить среднее время наработки до отказа (*MTTF*), воспользовавшись тем, что на анализируемом участке $\mu(t)$ является медленно убывающей функций и с достаточной степенью точности ее можно считать постоянной до момента обнаружения следующей ошибки: $\mu(t) = \mu(T_0)$, где T_0 – время начала эксплуатации, откуда $MTTF = n(T_0)\mu(T_0)$. На основе предоставляемой журналом системы управления версиями информации также можно определить среднее время ремонта (*MTTR*) и оценить коэффициент готовности разрабатываемого программного комплекса $A = MTTF / (MTTF + MTTR)$, а также время, необходимое для поиска и исправления ошибок с целью достижения требуемых эксплуатационных характеристик.

Необходимо отметить, что изменение текста программы в процессе тестирования не обязательно означает исправление обнаруженной критической ошибки: практика показывает, что заметная часть исправлений даже на этом этапе связана не столько с функциональностью или надежностью, сколько с изменениями пользовательского интерфейса, поэтому при оценке полученные значения *A* и *MTTF* должны быть нормализованы с учетом соответствующих пропорций. Требуемые коэффициенты могут быть получены также автоматически из журнала системы управления версиями при соблюдении разработчиками общих соглашений по оформлению комментариев (так, в рассматриваемом примере исправленные ошибки, которые могли повлиять на надежность и безопасность системы управления, помечаются модификатором «*[CRITICAL]*»).

Предложенный метод позволяет получить числовые оценки основных характеристик надежности программного обеспечения, таких как количество невыявленных ошибок, среднее время наработки на отказ, коэффициент готовности системы, вероятность опасного отказа за интересующий период времени и т. д., по информации, сохраняемой системой управления версиями.

Использование варианта трехзначной логики

При разработке систем управления сложными объектами зачастую возникает необходимость обеспечить предсказуемо-безопасное поведение в условиях поступающей неполной или недостоверной информации. В первую очередь это относится к распределённым системам с большим количеством входных датчиков, связь с которыми может быть потеряна, а сами датчики — выйти из строя. При обработке подобных ситуаций система, очевидно, должна различным образом реагировать на достоверную информацию (например, «напряжение ниже 220 В», «напряжение больше или равно 220 В») и ситуацию «отсутствует связь с датчиком напряжения». Если с технической стороны диагностику поступающей информации можно организовать многими хорошо известными и отработанными способами (такими как установка дублирующих датчиков и сравнение показаний, смещение нуля измеряемого тока или напряжения для определения обрыва провода и т. п.), то для корректной обработки необходимо использование специальных математических методов, позволяющих наряду с точными данными оперировать неполной или недостоверной информацией.

В основу построения таблиц истинности операторов предложенного варианта трёхзначной логики, представляющего собой расширение классической бинарной логики дополнительным истинностным значением «не определено» (или «неизвестно»), положены следующие принципы:

- сводимость к классической двузначной логике в случае, когда все аргументы функции принимают значения только «истина» (*t*) или «ложь» (*f*)
  ($F_{ternary}(x_1, x_2, \ldots) \equiv F_{binary}(x_1, x_2, \ldots)$, если все $x_i \in \{t, f\}$);

- если аргумент функции «неизвестен» (*u*) и при его изменении как на «истину», так и на «ложь» значение функции не изменяется (C), то результат вычисления принимается равным данной константе (если F (. . . , *t*, . . .) ≡ F (. . . , *f*, . . .) ≡ C, то F (. . . , *u*, . . .) ≡ C), в противном случае результат вычисления также «неизвестен» (F (. . . , *u*, . . .) ≡ *u*).

На основе этих принципов получены таблицы истинности логических операторов (табл. 1)

Табл. 1. Основные логические операторы

x	f	f	f	u	u	u	t	t	t
y	f	u	t	f	u	t	f	u	t
$\neg x$	t			u			f		
$x \wedge y$	f	f	f	f	u	u	f	u	t
$x \vee y$	f	u	t	u	u	t	t	t	t
$x \rightarrow y$	t	t	t	u	u	t	f	u	t
$x \equiv y$	t	u	f	u	u	u	f	u	t

Полученные таблицы истинности основных логических операторов представляют собой основную часть математического описания предложенного метода обработки данных на основе варианта трёхзначной логики, представляющего собой расширение классической бинарной логики дополнительным истинностным значением, которое интерпретируется как отсутствие в данный момент информации о точном значении переменной или нарушение одного из заданных ограничений значения переменной во время вычисления выражения.

Преимуществом данного метода является то, что он позволяет наряду с точными данными оперировать неполной и недостоверной информацией, существенно упрощая при этом запись логических выражений.

Использование варианта трехзначной логики [9] позволяет снизить сложность программного обеспечения и тем самым добиться повышения надёжности, что особенно актуально при создании автоматизированных систем управления, насчитывающих значительное количество входных сигналов, а поступающая информация обрабатывается логическими условиями со множеством аргументов. При этом программная реализация обеспечивает механизм автоматического переключения на работу в безопасном режиме тех подзадач, для корректного выполнения которых в данных момент недостаточно информации, а также позволяет локализовать некоторые типичные ошибки программирования и спецификаций аппаратной части.

Практическое применение метода, основанного на использовании варианта трёхзначной логики и унифицирующего обработку наряду с точными данными неполной и недостоверной информации, позволяет упростить разработку программного обеспечения, а также обеспечить корректное поведение и повысить надёжность всей системы управления.

Заключение

Предложенные в работе решения были использованы при построении автоматизированной системы диспетчерского управления движением поездов Новосибирского метрополитена. При этом новая система позволила не только увеличить надежность программ, но и значительно сократить время разработки. Более чем пятилетний опыт эксплуатации подтвердил эффективность решений, принятых при ее создании. На данный момент работы по модернизации станций метрополитена полностью завешены.

Литература

1. Надежность в технике. Основные понятия. Термины и определения. ГОСТ 27.002–89. — Москва: Государственный Комитет СССР по управлению качеством продукции и стандартам. Издательство стандартов, 1990.
2. *Димаки А.В.* Интегрированные системы проектирования и управления. — Томск: Кафедра информационно-измерительной техники, 2005. — 183 с.
3. *Конотоп Д.Г., Стефанюк А.Р., Яхно В.П.* Механизм отложенной передачи в системах диспетчерского контроля и управления // Промышленные контроллеры. АСУ. — 2008. — Т. 7. — С. 4–10.
4. *Белоконь С.А., Васильев В.В., Золотухин Ю.Н., Мальцев А.С., Соболев М.А., Филиппов М.Н., Ян А.П.* Автоматизированные системы диспетчерского управления объектами повышенной опасности // Автометрия. 2011. 47, №3. С. 73-83.
5. *Филиппов М.Н.* Разработка и исследование моделей и методов построения автоматизированных систем диспетчерского управления. Автореферат диссертации на соискание ученой степени кандидата технических наук // Типография Института катализа им. Г. К. Борескова СО РАН, 18 с.
6. *Гарбук С.В., Комаров А.А., Салов Е.И.* Аналитический отчет. Обзор инцидентов информационной безопасности АСУ ТП зарубежных государств (по материалам Интернет-изданий за 2008-2010 гг.) // http://www.securitylab.ru/analytics/398184.php
7. *Белоконь С.А., Филиппов М.Н.* Метод построения многоплатформенной открытой модульной SCADA-системы // Вест. НГУ. Сер. Физика. 2008. 3, вып. 1. С. 115–125.
8. *Филиппов М.Н.* Метод оценки надежности программного обеспечения по информации, сохраняемой системой управления версиями // Тр. XII Междунар. конф. «Проблемы управления и моделирования в сложных системах». Самара, Россия. ИПУСС РАН, 2010. С. 244–249.
9. *Филиппов М.Н.* Метод обработки неполных данных на основе трехзначной логики // Автометрия. 2009. 45, №5. С. 124-131.