ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И ПРИКЛАДНАЯ МАТЕМАТИКА

УДК 681.5 (075.8)

В.В. Белов, А.Н. Мыськин

КОНЦЕПЦИЯ ИНТЕЛЛЕКТУАЛЬНОЙ СИСТЕМЫ АНАЛИЗА СОСТОЯНИЯ ПРОГРАММНЫХ ПРОЕКТОВ

Предложена концепция интеллектуальной системы, предназначенной для анализа состояний программных проектов, идентификации рисков, определения оптимальных корректирующих мероприятий на основании прогнозов контролируемых процессов (метрик), включающей онтологические модели методов прогнозирования, метрик проектов, рисков, методологий управления проектами и опыта их применения на практике для унификации подхода к сбору и представлению знаний в области информационных технологий. Оценена целесообразность использования предложенной системы для управления проектами в современных IT компаниях.

Ключевые слова: *онтологии, прогнозирование, управление проектами, метрики.*

Введение. Информационные технологии получают все более широкое применение в современном мире. С каждым годом объем рынка информационных услуг возрастает, сложность разрабатываемых систем увеличивается. Вместе с ростом объемов и сложности информационных систем ужесточаются требования к качеству выпускаемых продуктов. При этом наиболее критичной становится проблема управления разработкой программного обеспечения (ПО) в информационно-технологической (ИТ) компании. Современные проектные команды достигают сотен и более человек, непосредственно участвующих в производственном процессе. Создание программных систем становится технологией, где у каждого члена проектной команды определено свое место и круг обязанностей, где строго регламентированы все этапы — от замысла до передачи пользователям рабочей версии программы. Это единственный путь к созданию больших и малых программных систем, который позволяет уложиться в установленные сроки и выделенный бюджет, обеспечив при этом систему требуемого качества.

Цель данной работы состоит в разработке концепции построения интеллектуальной системы управления программными проектами, позволяющей снизить риски за счет постоянного мониторинга текущего состояния проекта, прогнозирования ключевых характеристик и заблаговременного принятия мер для повышения эффективности и качества процессов жизненного цикла проектов.

Предложенное в работе формализованное описание метаданных на основе онтологий позволяет осуществить унификацию процесса проектирования современных программных систем за счет снижения вероятности возникновения проектных рисков, оценки эффективности применяемых мероприятий, внесения корректировок в производственные процессы.

Проблемы: дефицит сроков, фондов и компетенций. Современные проекты сталкиваются с рядом сложностей: сжатые сроки разработки, сокращенные бюджеты, недостаточное количество квалифицированных ресурсов. В связи с этим руководителям проектов приходится постоянно модифицировать подходы к управлению. Мероприятия, вводимые на первом этапе разработки, могут в последствии оказаться мало эффективными: средства, затраченные на внедрение мероприятий, превосходят эффект от них. Каждый день руководитель проекта сталкивается со сложной задачей выбора правильного

решения. При этом в мире разрабатываются ежегодно тысячи проектов, использующих десятки методологий и сотни подходов. И при внедрении каждого подхода наблюдаются определенные результаты, которые могли бы быть использованы на других проектах, помогая выбрать правильный путь развития. Но, чтобы это стало возможным, необходимо выполнить ряд сложных задач.

1. Корректно оценить состояние текущего проекта, выявив специфичные характеристики используемых процессов.

2. Идентифицировать риски, которые могут наступить при текущем состоянии проекта.

3. Провести анализ проекта и осуществить прогноз дальнейшего развития проекта.

4. Выбрать подходы разработки и управления, которые позволят минимизировать возникновение рисков.

5. Провести мониторинг внедренных подходов, оценить их эффективность, определить корректирующие мероприятия.

При этом указанные задачи должны выполняться периодически: ситуация на проекте может изменяться очень быстро.

Стандарты и модели управления жизненным циклом ПО. Во всех современных стандартах и методологиях управления жизненным циклом программного обеспечения уделяется значительное внимание сбору и управлению метриками. Именно метрики позволяют количественно производить мониторинг хода проекта и оценить его текущее состояние.

Так, в методологии RUP (IBM Rational Unified Process) в рамках дисциплины *Управление проектом* выделяется задача *Мониторинг состояния проекта*. На выходе данной задачи составляется артефакт *Показатели проекта*, представляющий собой активное хранилище показателей проекта. RUP рассматривает метрики для различных аспектов проекта:

- ход разработки в форме размера и сложности;
- стабильность в форме частоты изменения требований или реализации, размера или сложности;
- модульность в форме области изменений; качество в форме числа и типа ошибок;
- зрелость в форме частоты появления новых ошибок;
- ресурсы в форме отношения фактических расходов к планируемым.

При этом RUP отмечает, что важнее отслеживать тенденции, чем абсолютные значения метрик.

Модель CMMI (Capability Maturity Model Integration) придает еще большее значение сбору и анализу метрик. Четвертый уровень носит название «Управляемый количественно» и одной из двух процессных областей данного уровня является Quantitative Project Management (количественное управление проектом). Кроме того, необходимость сбора метрик упомянута уже на втором уровне в процессной области Measurement and Analysis (измерение и анализ). Это означает, что начиная со второго уровня, компания должна собирать, анализировать и управлять метриками [4, 5].

В моделях ускоренной разработки ПО (Agile) процесс сбора и анализа метрик является также весьма существенным. Между выпусками необходимо контролировать основные показатели (ресурсы, качество, время), во время работы над выпуском следует уделять внимание оценке работы проектной команды, на протяжении всего проекта необходимо выполнять мониторинг результатов [1].

Поскольку концепция MSF (Microsoft Solutions Framework) основана на CMMI и Agile, в ней также большое внимание уделяется сбору и анализу метрик. Данный процесс должен выполняться на протяжении всего жизненного цикла ПО.

Ключевые понятия, термины и методы измерений в приложении к программному обеспечению определены в стандарте ISO 15939 “Software Engineering - Software Measurement Process” и международном словаре метрологии ISO. ISO 15939 также определяет стандартный процесс для измерения характеристик процессов и продуктов.

Метрики в управлении проектами и инструменты их измерения. Метрики программных проектов – это количественные показатели, отражающие их отдельные характеристики; измерение или комбинация измерений, выполняемые внутри программного проекта или процесса [2].

Измерения могут проводиться:

- для поддержки, инициирования, реализации и изменения процессов и для оценки результатов этих процессов;
- для оценки результатов процессов – рабочих продуктов, включая программное обеспечение и документацию;
- для оценки проектов: совокупности ресурсов, задач и результатов работы проекта;
- для оценки ресурсов: работников, методов и утилит, времени, трудозатрат и бюджета в распоряжении проекта.

Измерения могут производиться в отношении процессов и в отношении непосредственно разрабатываемых продуктов. Измерения в отношении процесса подразумевают сбор,

анализ и интерпретацию количественной информации о процессе. Измерения используются для идентификации сильных и слабых сторон процесса и для оценки процесса после того, как он реализован и/или изменен. Это очень важный аспект использования метрик, поскольку метрики позволяют делать выводы об успешности внедрения тех или иных мероприятий на проекте. Измерения в отношении программного продукта включают количественную оценку его размера, структуры и качества.

Сбор и анализ метрик является весьма затруднительной и дорогостоящей задачей без использования специализированных систем. Метрики должны разрабатываться в соответствии с требованиями клиентов и организации, тестироваться на предмет возможности реализации, должен выполняться постоянный мониторинг эффективности выбранных показателей. Однако в связи с существенными различиями в характере и назначении собираемых показателей достаточно сложно разработать единую систему управления метриками. Чаще всего сбор различных групп метрик привязывают к специализированным инструментам. Сбор метрик, связанных с разработкой и получением кода, обычно осуществляют с помощью систем контроля версий. Типичными представителями таких систем являются Subversion и IBM Rational ClearCase. Эти системы позволяют управлять версиями разрабатываемого решения и снабжать их необходимыми атрибутами. В атрибутах как раз и осуществляется хранение показателей для дальнейшего анализа. В системах контроля версий можно осуществлять хранение таких метрик, как количество строк кода, количество добавленных комментариев, сложность написанного кода и другие показатели, связанные непосредственно с кодом.

Метрики, связанные с дефектами и запросами на улучшение программного решения, получают, как правило, с помощью систем багтрекинга (Bugzilla, IBM Rational ClearQuest, RedMine). Данные системы предназначены для управления, отслеживания и хранения ошибок и запросов на расширение функциональности программного продукта. Информацию о ресурсах проекта и о финансовой составляющей получают из специализированных систем планирования (Microsoft Project, Rational Portfolio Manager).

Отдельно следует сказать о крупных программных комплексах, позволяющих собирать и управлять метриками из нескольких групп. Здесь можно выделить два лидера: система Microsoft Visual Studio Team System и группа продуктов Rational компании IBM. Данные комплексы представляют собой интегрированное решение для управления жизненным циклом приложений, в состав которого входят программные средства, процессы и руководства. В системах используется компонентный принцип построения программного обеспечения с разделением инструментов по ролям и специализациям пользователей.

Все перечисленные системы имеют значительную стоимость и не позволяют решать задачу управления метриками комплексно. Наиболее дорогие системы способны автоматизировать сбор метрик в различных направлениях, но производить анализ метрик и, тем более, предлагать корректирующие мероприятия они не могут.

В настоящее время основной проблемой количественного управления проектом является неоднозначность и сложность выбора показателей для отслеживания.

Выбор между минимальным и более широким набором показателей делается на основе параметров проекта и ряда других условий. Необходимо учитывать, например, размер проекта, повышенные требования к безопасности и надежности, сбор определенных показателей может требоваться контрактом или руководством компании в целях поиска способов улучшения в определенных областях. При этом не существует универсального решения и рекомендаций для выбора метрик, и руководитель проекта должен самостоятельно выбрать подходящий набор показателей. Ориентируясь на собственную интуицию и опыт, сделать правильный выбор очень непросто.

Здесь существенным подспорьем могла бы стать информация об опыте использования тех или иных метрик в близких по уровню сложности, размеру и тематике проектах. Однако наличие подобных проектов и, тем более, зафиксированных данных о проводимых мероприятиях является весьма редким явлением. Руководитель проекта ориентируется на свой личный опыт работы или, в лучшем случае, на опыт «соседних» проектов внутри своей компании. Очевидно, что на основании только этой информации сделать правильные выводы об опыте использования метрик затруднительно.

Другой проблемой является сложность интерпретации результатов и использование полученных данных для предвидения ситуации на проекте. Мало получить необходимые статистические данные, необходимо провести корректно анализ этих данных и понять, какие мероприятия следует проводить для изменения

ситуации на проекте. Современные системы позволяют лишь собирать и накапливать статистические данные о ходе проекта, практически не помогая провести анализ ситуации. Здесь снова на передний план выходит человеческий фактор.

Разный характер измерений приводит к необходимости использования отдельных инструментов управления проектом. Другими словами, для получения исчерпывающей информации о проекте приходится использовать несколько несвязанных или слабо связанных инструментов. Отсутствие централизованной системы управления метриками существенно увеличивает стоимость мероприятий, связанных с анализом.

Наконец, процесс сбора и анализа метрик не должен существенно усложнять и удорожать разработку проекта. Сейчас для сбора и, главное, интерпретации результатов требуются существенные дорогостоящие ресурсы. Принцип экономичности управления требует: затраты на получение информации не должны превышать эффект, получаемый от её использования. В связи с этим от задачи статистического управления проектом зачастую отказываются в пользу классических подходов.

Пути решения проблем управления проектами. С учетом вышеизложенного следует, что необходимо:

- разработать единую централизованную систему управления метриками, использующую универсальный аппарат накопления и представления данных;
- сделать возможным взаимодействие проектов не только внутри одной организации, но и между компаниями-партнерами. Данное взаимодействие должно выражаться в реализации единой системы знаний о практиках использования метрик и их полезности для повышения эффективности проекта;
- создать условия для свободного расширения системы: необходима возможность введения новых предметных областей, характеристик проектов и метрик;
- предоставить поддержку аналитикам и руководителям проектов в интерпретации статистических данных, полученных в ходе мониторинга проектов. Исследователь должен иметь возможность не только анализировать эффективность проводимых мероприятий по историческому опыту предшествующих проектов, но и получить возможность предвидения ситуации на текущем проекте.

Следует заметить, что указанные меры не принижают роль исследователя и руководителя, а лишь помогают ему принять правильное решение для реализации проектных мероприятий.

Сам по себе сбор метрик практически бесполезен: необходимо корректно интерпретировать результаты. Имея определенный набор показателей проекта, можно идентифицировать проектные риски. При этом для идентификации той или иной проблемы может быть недостаточно одной метрики, необходимо анализировать сразу несколько показателей. Данная задача сама по себе является непростой. Для ее решения необходим аппарат анализа входных данных и система логического вывода, позволяющая по исходным фактам сделать выводы о возможных рисках. Кроме того, возможные риски должны быть классифицированы и снабжены рядом признаков для их более точной идентификации.

Проанализировав метрики и определив риски, возникает задача поиска оптимальных методов управления и корректирующих проектных мероприятий. Их выбор должен осуществляться на основании данных о предметной области проекта, его специфике, выбранной методологии разработки. При этом следует учитывать, что к настоящему моменту разработаны десятки методологий управления проектов, сотни полезных практик.

После определения и внедрения корректирующих мероприятий необходимо проводить мониторинг ситуации на проекте и оценивать эффективность принятых мер. Далее этапы, описанные ранее, повторяются.

Важной задачей является предвидение хода развития проекта. Здесь может помочь прогнозирование метрик. При этом следует понимать, что прогнозирование – отдельная задача, требующая особого внимания. На качество выполненного прогнозирования влияет ряд факторов: выбор метода прогнозирования, подход к определению свободных параметров, объем и качество исходных данных и т.п. Исследования в данной области ведутся одновременно по нескольким направлениям: регрессионные и авторегрессионные модели, нейронные сети. По каждому из направлений создано множество методов, предложены подходы для определения оптимальной структуры прогнозирующих описаний. Возникает задача в максимальной степени автоматизировать процесс прогнозирования. Для этого необходимо единое описание всех имеющихся методов и полученных практических результатов.

Онтологии, как средство формализации знаний в системах управления программными проектами. Для унификации подхода к сбору и представлению знаний предлагается

использовать онтологии. Онтология (в информатике) – это попытка всеобъемлющей и детальной формализации некоторой области знаний с помощью концептуальной схемы [3]. Обычно такая схема состоит из иерархической структуры данных (таксономии), содержащей классы объектов, их связи и правила (теоремы, ограничения), принятые в этой области. Одно из основных преимуществ использования онтологий – возможность объединения информации, полученной из различных информационных сообществ. Языки описания онтологий позволяют обеспечить взаимодействие между множеством независимо развивавшихся схем представления тех или иных знаний. Кроме того, онтология – конструкция, удобная для представления знаний в Интернете, что является важным для развития системы (в определенном смысле, языки описания онтологий являются продолжением развития технологии XML).

Использование онтологий позволит создать единую базу знаний, содержащую информацию о проектах, в частности о метриках и опыте их использования. Более того, опираясь на сущность онтологий, как средства преставления и хранения данных, система хранения информации о проектах может быть расширена за счет опыта других компаний. Это означает, что при старте очередного проекта, можно принимать решения, основываясь не только на своем опыте, но и на опыте коллег.

Состав онтологий интеллектуальной системы управления программными проектами. Рассмотрим основные онтологии, используемые в концепции описываемой системы.

Онтология метрик. Корневыми классами в таксономии будут выступать области применения метрик: планирование и прогресс, ресурсы и стоимость, стабильность и размер продукта, качество продукта, эффективность процессов, удовлетворенность заказчика. В качестве экземпляров - непосредственно метрики: время ОТК-лика, сходимость дефектов, трудозатраты на релиз, плотность дефектов, стабильность требований и т.п. Основными свойствами метрик будут являться: список первичных показателей, рекомендуемые целевые и критические пороги, рекомендуемая периодичность съема показаний, объемы проектов, для которых использование метрик целесообразно. Данная онтология связана с базой данных проектов через базу данных прецедентов использования метрик, определяя факты использования метрик на конкретных проектах.

Онтология рисков. Базовые классы данной онтологии – категории рисков: технические, внешние, риски окружающей среды и управления проектами, риски тестирования. Экземпляры – непосредственно риски. Онтология рисков через базу данных истории рисков связана с базой данных проектов, определяя, какие риски проявлялись на каких проектах.

Онтология методов управления проектами и корректирующих мероприятий. Базовые классы – методологии разработки проектов: RUP, MSF, Agile, CMMI. Экземпляры – конечные практики методологий (например, парное программирование, unit тестирование, тестирование на территории заказчика, мониторинг решения аналитиком и т.п.). Онтология связана с базой данных проектов, а также онтологией рисков через базу данных истории рисков.

Онтология методов прогнозирования. Корневыми классами в таксономии выступают основные направления исследований (все виды авторегрессий, оптимальные полиномы, искусственные нейронные сети, чёткие и нечёткие рекуррентные схемы, взвешивающие схемы, фрактальные схемы и т.д.), а в качестве экземпляров – конкретные методы прогнозирования (например, модель авторегрессионного интегрированного скользящего среднего ARIMA, метод Винтерса, модель Бурга, конкретный вид и параметры нейронной сети и т.д.). В описании методов могут входить входные и выходные параметры, наборы свободных переменных, а также способы их определения, рекомендации для использования методов в определенных предметных областях и требования, предъявляемые к исходным данным.

Онтология методов прогнозирования связана с базой данных проектов через специальную связующую базу данных *Прецеденты прогнозирования*. Под прецедентом прогнозирования в данном случае понимается совокупность исходных данных конкретного процесса, результатов прогнозирования, оценок качества полученного прогноза, а со временем фактические данные и результаты их сравнения с прогнозными значениями.

На основании собранных метрик, а также на основании полученных прогнозов метрик могут быть сформулированы рекомендации по внедряемым корректирующим мероприятиям и методам управления.

На рисунке 1 показана взаимосвязь онтологий рассматриваемой системы.

Центральной базой данных является база проектов. Каждый проект связан с некоторой предметной областью. Возможные предметные области содержатся в онтологии «Предметные области». Возможные метрики, а также их

характеристики описываются онтологией «Метрики». Информация о фактах использования метрик на проектах содержится в базе данных «Прецеденты использования метрик». Прогнозирование метрик осуществляется с помощью методологий, описываемых в онтологии «Методы прогнозирования». Факты использования методов прогнозирования содержатся в базе данных «Прецеденты использования методов прогнозирования». Непосредственно факты прогнозирования, прогнозные и фактические значения показателей содержатся в базе данных «Прецеденты прогнозирования». На основании собранных и спрогнозированных метрик осуществляется идентификация рисков (онтология «Риски», база данных «История рисков»), поиск методов управления проектом и корректирующих мероприятий (онтология «Методы управления проектами и корректирующие мероприятия», база данных «Прецеденты использования методов управления и корректирующих мероприятий»). История возникновения рисков хранится в базе данных «История рисков». Кроме того, осуществляется коррекция алгоритма прогнозирования.

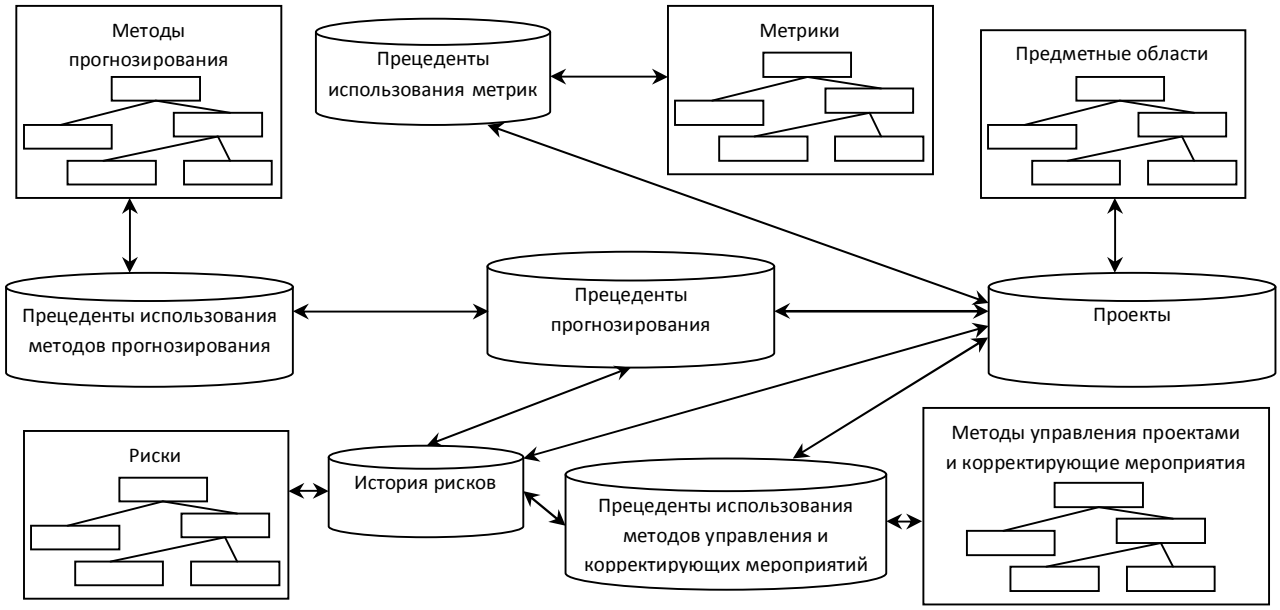

Рисунок 1 - Связь онтологий и баз данных системы

Архитектура интеллектуальной системы управления программными проектами. На рисунке 2 представлена схема реализации интеллектуальной системы. В качестве СУБД для хранения онтологий может быть использована MySQL, Oracle, PostgreSQL, MS SQL. Конкретная СУБД определяется возможностями используемых средств управления онтологиями. Управление онтологиями может осуществляться с помощью систем для работы с онтологиями (Protege) или с помощью специализированных библиотек (Jena). Специализированные системы позволяют определять классы, экземпляры, свойства онтологий, используя средства пользовательского интерфейса. Разработанная онтология может храниться во внешних файлах или заданной СУБД.

Библиотека Jena, разработанная HP Labs, написана на Java и позволяет с помощью API интерфейса осуществлять доступ с OWL онтологиями, выполнять SPARQL запросы к онтологиям, использовать для хранения и работы с онтологиями СУБД. Модуль прогнозирования и модуль идентификации рисков могут быть реализованы с помощью средств языков программирования высокого уровня или с помощью специализированных математических пакетов.

Клиентское приложение, с помощью которого предполагается управление системой, реализуется на любом языке высокого уровня. С помощью данного приложения конечные пользователи будут осуществлять управление системой: добавлять новые прецеденты использования метрик, прецеденты прогнозирования, формулировать запросы к системе, получать рекомендации о корректирующих мероприятиях и т.п. Фактически, доступ к ядру системы может осуществляться либо с помощью клиентского приложения, либо с помощью систем управления онтологиями.

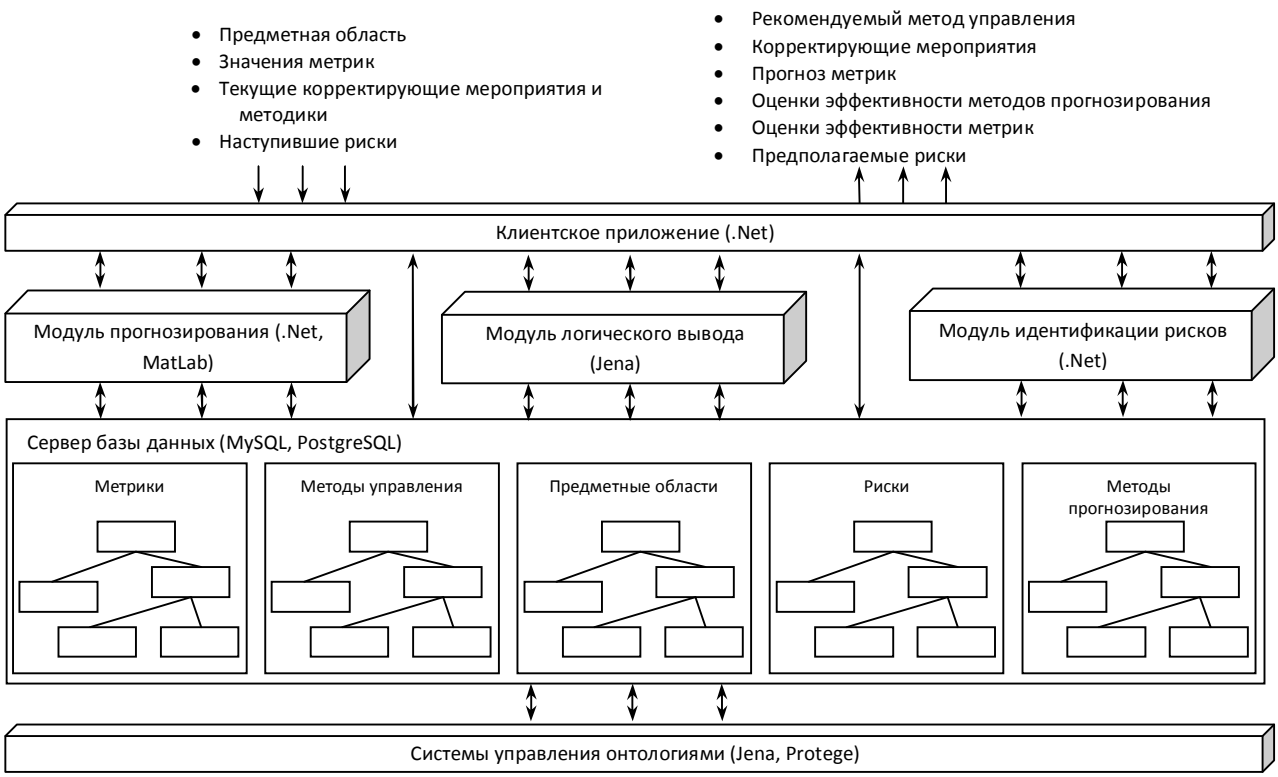

Рисунок 2 - Структура интеллектуальной системы анализа проектов

Таким образом, передавая на вход системы информацию о предметной области, значения метрик, список текущих корректирующих мероприятий и методик, список рисков, можно получить следующую информацию.

1. Прогноз метрик, выполненный с помощью автоматически определенного метода прогнозирования.

2. Список предполагаемых рисков, полученный на основании выполненного прогноза и полученных исходных данных.

3. Рекомендуемый метод управления, сформированный на основании исходных данных и списка рисков.

4. Рекомендуемые корректирующие мероприятия.

5. Оценки эффективности метрик.

6. Оценки эффективности методов прогнозирования.

Заключение. Научным результатом, представленным в данной работе, является концепция интеллектуальной системы, предназначенной для анализа состояний программных проектов, идентификации рисков, определения оптимальных корректирующих мероприятий на основании прогнозов контролируемых процессов. Отслеживание состояния проектов осуществляется с помощью метрик, выбор которых и последующая корректировка осуществляется с помощью предложенной системы. Решение о внедряемых мероприятиях формируется на основе реальных и прогнозных значений метрик, получаемых с помощью набора методов прогнозирования.

Для унификации подхода к сбору и представлению знаний предложено использовать онтологии. Разработаны онтологические модели методов прогнозирования, метрик проектов, рисков, методологий управления проектами и опыта их применения на практике. Предложены схема взаимосвязи онтологий, схема построения интеллектуальной системы.

Предложенная концепция интеллектуальной системы может быть использована в любых организациях, занимающихся промышленной разработкой программного обеспечения. Основным преимуществом системы является возможность разработки отдельных модулей системы, их расширения, наполнения баз данных различными группами специалистов. Например, руководители проектов осуществляют наполнение базы знаний методов управления и рисков, основываясь на опыте внедрения мероприятий на проектах, математики и аналитики описывают новые математические методы и подходы прогнозирования в базе данных и онтологии методов прогнозирования. Система имеет высокую степень масштабируемости: каждая онтология может быть дополнена независимыми научными сообществами и проектными группами.

Сложность работы с системой минимизируется за счет существующих инструментов доступа к базе знаний на основе онтологий. Экономический эффект от внедрения системы будет достигнут за счет минимизации случаев неэффективного управления проектами, а также за счет сокращения времени работы сотрудников с узкой специализацией в области аналитики, математических методов, статистики и прогнозирования.

В настоящее время осуществляется спецификация и разработка прототипа системы.

Библиографический список

1. *Бек К.* Экстремальное программирование. - СПб.: Питер, 2002. – 224 с.: ил. ISBN: 5-94723-032-1

2. *Брукс П.* Метрики для управления ИТ-услугами. Пер. с англ. – М.: Альпина Бизнес Букс, 2008.

3. *Клещев А.С., Артемьева И.Л.* Математические модели онтологий предметных областей. Часть 1. Существующие подходы к определению понятия «онтология» // Науч.-техн. информ. Сер. 2, Информ. процессы и системы. – 2001.

4. *Мацяшек, Лешек А.* Анализ и проектирование информационных систем с помощью UML 2.0, 3-е изд.: пер. с англ. – М.: И.Д. Вильямс, 2008.

5. *Ройс У.* Управление проектами по созданию программного обеспечения. Пер. с англ. – М.: Лори, 2007. ISBN 5-85582-156-0.

УДК 381.322

А.П. Мартынов

ОБЕСПЕЧЕНИЕ БЕЗОПАСНОГО ФУНКЦИОНИРОВАНИЯ ПРОГРАММНО-АППАРАТНЫХ СРЕДСТВ В ПРОЦЕССЕ ИХ ИНИЦИАЛИЗАЦИИ

Представлены варианты структурного построения загрузчиков программно-математического обеспечения, на их основе предложен универсальный загрузчик и даны практические рекомендации по его использованию для обеспечения надежного функционирования программно-аппаратных средств.

Ключевые слова: *загрузчик, конфигурация системы, преобразование данных, операционная система.*

Введение. Среда MS-DOS является наиболее изученной операционной системой в нашей стране. Что же касается среды MS-WINDOWS, то информация по организации её работы многогранна. Автор программы для среды MS-WINDOWS не претендует на абсолютное обеспечение безопасности исполняемых файлов, но надеется, что применение данного загрузчика позволит в определенной степени обеспечить безопасное функционирование применяемых программ.

Цель работы – обеспечение надежного функционирования программно-аппаратных средств за счет повышения безопасности программно-математического обеспечения.

Рассмотренный загрузчик позволяет запускать на выполнение любые исполняемые файлы из командной строки, а также его вызов может осуществляться из другого файла.

Основным требованием, предъявляемым к обеспечению безопасности исполняемых файлов, является невозможность их нелегального использования, а именно:

1) невозможность нелегального запуска исполняемых файлов на выполнение;

2) невозможность нелегального копирования исполняемых файлов;

3) защиту исполняемых файлов от заражения вирусом.

Кроме того, исполняемый файл должен быть защищён от действий дизассемблера, отладчика и трассировки по заданному прерыванию. Все эти требования должны учитываться в процессе создания и эксплуатации автоматизированных систем.

Обеспечить безопасность исполняемого файла от нелегального использования можно двумя основными способами:

1) вставкой фрагмента проверочного кода в исполняемый файл;

2) преобразованием исполняемого файла к неисполняемому виду и применением для загрузки не операционной среды, а некоторой программы, в теле которой и осуществляются необходимые проверки и преобразования.

Разработка загрузчика исполняемых файлов. MS-DOS поддерживает два типа исполняемых файлов: COM и EXE. Файлы типа COM – более простые в смысле обеспечения безопасности, они являются всего лишь двоичным образом задачи. При запуске такого файла MS-DOS считывает его в память по смещению 100h и передаёт управление на его первый байт. В том же сегменте по смещению 0 записывается PSP (префикс программного сегмента) – структура данных, необходимая программе для получения доступа к элементам командной строки и др. Файлы типа EXE более сложны по своей структуре. Эти файлы предназначены для создания программ, код или данные которых превышают по размеру 64 Кб (максимальный размер COM файла). Так как максимальный размер сегмента в среде MS-DOS также равен 64 Кб (что является ограничением на длину COM файла), то MS-DOS записывает коды и данные из EXE файла в несколько сегментов, используя информацию из структуры, находящейся в начале файла (из заголовка EXE файла). Файл EXE состоит из двух частей: управляющей информации для загрузчика и собственно загрузочного модуля. Информация для загрузчика расположена в начале файла и образует заголовок. За ним следует тело загрузочного модуля. Тело загрузочного модуля начинается на границе блока и представляет собой копию образа памяти задачи. Сложность обеспечения безопасности этого типа программ состоит в том, что кроме модификации кода и добавления кода безопасности к файлу приходится приводить данные в заголовке EXE файла в соответствие с внесенными в него изменениями для обеспечения правильной загрузки и выполнения основного кода после выполнения кода безопасности.

Необходимость дополнения готового исполняемого модуля некоторыми функциями, которые выполняются до начала работы основного модуля, встречается достаточно часто. Это может быть, например, просчёт контрольных сумм с целью обнаружения файлового вируса, установление парольного доступа к файлу или проверка наличия некоторой идентифицирующей информации (например, для защиты от копирования). Модификация программы с целью добавления необходимого программного кода безопасности реализуется вирусным методом. Для программы в формате EXE файла в конец файла добавляется необходимый код, корректируются точка входа и размер файла. Для программ в формате COM файла (длина результирующего файла не превышает 64 Кб) требуемый код добавляется в конец файла, а часть кода, находящегося в начале программы, корректируется для безусловного перехода на добавленный код. Получив управление, добавленный код должен выполнить аутентификацию пользователя, персонального компьютера или дискеты. Эту функцию следует выполнять в закрытом от изучения коде. При положительном результате аутентификации управление передаётся основной программе. В противном случае выполнение программы прекращается и может быть, например, выдано сообщение о нелегальном действии.

Большинство программ загружаются в память, запускаются, а затем удаляются из памяти операционной системой. Если же для обеспечения безопасности программы от нелегального запуска преобразовать загрузочный модуль, то MS-DOS не сможет выполнить загрузку. В этом случае загружаемая программа будет защищена от заражения вирусом, дисассемблирования и отладчика, копирование преобразованного файла также не приведет к желаемому результату.

Для загрузки программ MS-DOS использует функцию 4Bh прерывания int 21h, предполагая, что загрузочный модуль имеет определённую структуру. Естественно в преобразованном модуле эта структура не соблюдается, и попытка загрузить такую программу окончится неудачей. Для запуска преобразованных программ нужен загрузчик (LOADER), который будет считывать преобразованную программу в память, осуществлять ее восстановление, выполнять работу MS-DOS по настройке адресов и заполнению служебных полей и после этого передавать ей управление. LOADER может выполнять проверку легальности запуска преобразованных программ. После запуска программы, сам загрузчик может “исчезать” из памяти, чтобы не отнимать ресурсы у программы. Локализация проверочных кодов в загрузчике позволяет избежать увеличения размеров преобразованных программ. Если же загрузчик не копировать на жесткий диск и всегда запускать его с персонального носителя данных, который хранится у владельца преобразованных программ, то получить не преобразованную копию программ не сможет даже самый опытный “хакер”. В этом случае носитель с загрузчиком будет играть роль физического ключа для восстановления программ.

Трассировка программы по заданному прерыванию заключается в следующем: “программа-шпион” адресует вектор заданного прерыва-

ния на собственную функцию, которая до или после выполнения прерывания останавливает процесс выполнения программы и позволяет “программе-шпиону” ознакомиться с интересующей информацией.

Для обеспечения безопасности от подобных действий можно предпринять следующие меры:

1) не использовать прерывания в загрузчике;

2) блокировать трассировку прерываний int 13h, int 40h, int 21h;

3) после отработки существенно значимого фрагмента кода приводить его в нерабочее состояние.

Для обеспечения безопасности загрузчика должно быть исключено попадание его носителя посторонним лицам, а запуск загрузчика с носителя необходимо осуществлять, указав в командной строке путь к преобразованному исполняемому файлу. Например: a:\loader c:\work\filename

Разработка программного обеспечения запуска исполняемых файлов проводилась по двум направлениям:

1) разработка загрузчика исполняемых файлов в среде MS-DOS;

2) разработка загрузчика исполняемых файлов в среде MS-WINDOWS.

Блок-схема программы LOADER в среде MS-DOS приведена на рисунке 1. Эту программу можно использовать и в среде MS-WINDOWS, для запуска файлов, созданных для работы в среде MS-DOS. В этом случае MS-WINDOWS открывает MS-DOS-процесс для выполнения этих файлов.

Преобразование исполняемого файла может осуществляться по одному из известных алгоритмов преобразования, например ГОСТ 28147-89, при этом конфиденциальные параметры преобразования следует расположить на носителе вместе с загрузчиком, в преобразованном виде или получать их только в процессе работы, например с внешнего устройства, через последовательный порт. Для носителя можно использовать нестандартное форматирование или иные способы достижения безопасности. Опасным моментом является легальное считывание конфиденциальных параметров с носителя. Для этого используется прерывание int 13h. Для обеспечения защиты от возможного его перехвата следует заменить данное прерывание на прямое обращение к BIOS.

LOADER защищён от трассировки отладчиком с помощью измерения времени выполнения участка программы. Также, если работа программы контролируется отладчиком, то клавиатура персонального компьютера будет блокирована. Что же касается защиты от дизассемблера, то организовать надёжную защиту загрузчика достаточно сложно и она в данной работе не затрагивается. В начале выполнения программы LOADER производится замер времени выполнения эталонного участка программы. Для получения отсчётов времени используется системный счётчик 0000h:046Ch. Если программа работает под управлением отладчика, то происходит выход (можно также в этом случае генерировать коды с помощью счётчика случайных чисел). Иначе происходит переход на процедуру func, где проводится проверка правильности ввода с командной строки. Если при вводе обнаружена ошибка, то происходит выход из программы. Далее, проводится подготовка среды (иногда применяется термин “окружение” – ENVIRONMENT) для запускаемой программы. Окружение MS-DOS – это область памяти длиной до 32 Кб, в которой MS-DOS сохраняет переменные окружения COMSPEC, PATH, PROMPT и т.п. Каждая переменная окружения представляет собой текстовую строку. Каждая строка окружения оканчивается нулём, конец окружения обозначается двумя нулями подряд. Начиная с версии MS-DOS 3.0, после окружения располагается дополнительная строка, содержащая полное имя запущенной программы (с указанием диска и маршрута поиска) и, возможно, параметры обращения к программе. Таким образом, чтобы найти имя запускаемого файла, нужно отыскать в окружении MS-DOS два ноля подряд – это признак расширенной части окружения. Слово, следующее за этим признаком, содержит количество переменных в расширенной части, за ним помещаются сами переменные. Например, в терминах ассемблера структура окружения может быть такой:

```
    db ‘COMPREC=C:\COMMAND.COM’,0 ;
Переменная COMSPEC
    db ‘PATH=C:\;C:\DOS’,0        ;
Переменная PATH
    db ‘PROMPT=‘$p$g’,0           ;
Переменная PROMPT
    dw 2                          ;
Количество переменных в расширенной части
    db ‘A:\LOADER’,0              ;
Имя файла
```

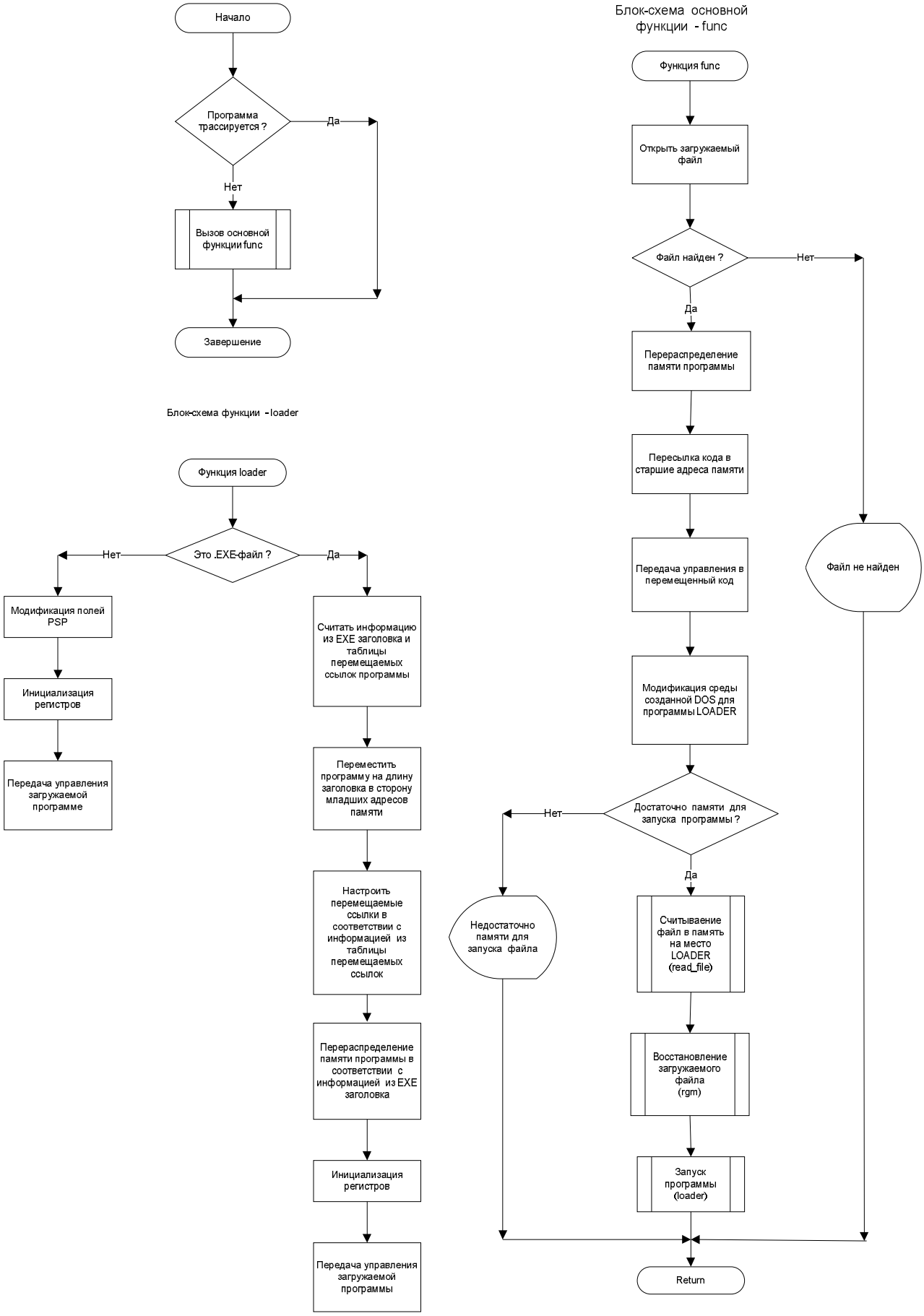

Рисунок 1 - Блок-схема программы LOADER в среде MS-DOS

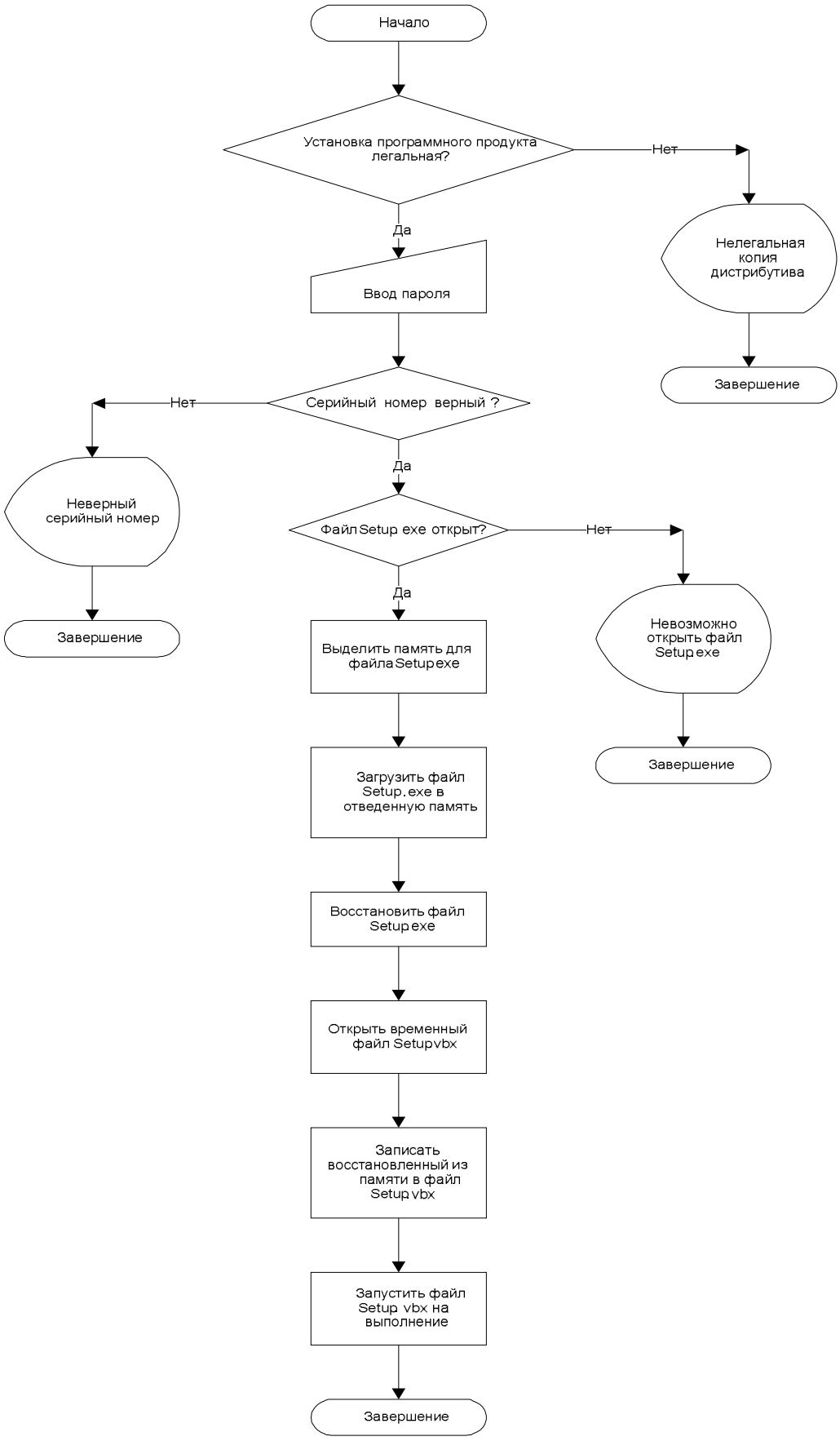

Рисунок 2 - Блок-схема программы LOADER в среде WINDOWS

Для каждой программы MS-DOS создаёт отдельную копию окружения. Сегментный адрес среды помещается в префикс программного сегмента (PSP). После запуска программы нет необходимости выделять новую память под окружение запускаемой программы, можно воспользоваться окружением, которое создало DOS для самого загрузчика при его запуске, так как сам загрузчик уже не нужен. Для этого достаточно после окружения загрузчика вписать имя запускаемой программы. Если подобным же образом повторно использовать и PSP загрузчик (для этого надо загружать запускаемую программу в то же место памяти, куда был загружен LOADER), то подготовку среды можно считать законченной, так как в PSP уже стоит правильный адрес окружения.

Имя может оказаться длиннее, чем полное имя загрузчика и не поместиться в область памяти окружения. В этом случае можно попробовать расширить блок памяти окружения или запросить пользователя скопировать LOADER в один каталог с запускаемой программой и ещё раз запустить его. При этом длины полных имён загрузчика и программы будут почти одинаковы. На втором этапе работы необходимо загрузить программу в память, причём на то же место, куда был загружен LOADER. Сначала надо освободить место под загрузку программы. Для этого можно переместить код загрузчика в старшие адреса памяти и передать туда управление. После этого надо определить длину загружаемого файла и записать её в поле длины области памяти блока MCB (блок управления памятью) того сегмента памяти, где изначально располагался загрузчик. Теперь можно считать программу в память, начиная со следующего после PSP загрузчика параграфа. На третьем этапе производится восстановление программы в памяти.

Алгоритм работы программного обеспечения. С учетом вышесказанного, алгоритм работы загрузчика будет следующий. Определяется формат загрузочного модуля: EXE или COM. Если программа имеет COM-формат, то достаточно заполнить некоторые поля PSP, соответствующим образом инициализировать регистры и передать управление загруженной программе с адреса CS:100h. С этого момента программа начнёт выполняться так, как если бы она была загружена MS-DOS. Если программа имеет формат EXE, то необходимо сделать несколько дополнительных шагов:

1) считать информацию из EXE-заголовка и таблицы перемещаемых ссылок программы;

2) переместить программу на длину заголовка в сторону младших адресов памяти;

3) настроить перемещаемые ссылки в соответствии с информацией из таблицы перемещаемых ссылок;

4) перераспределить память программы в соответствии с информацией из EXE-заголовка.

Только после этих шагов можно инициализировать регистры и передавать управление загруженной программе. Так как при пересылке загрузчика в старшие адреса адресного пространства никаких выделений памяти не производится, для MS-DOS это перемещение остаётся незамеченным, а значит MS-DOS будет считать всю память, расположенную после запускаемой программы, свободной. Таким образом, загрузчик как бы “исчезает” из памяти после выполнения своей функции. По окончании работы программы MS-DOS обычным образом либо очистит занимаемую программой память, либо оставит программу резидентной в зависимости от того, как она заканчивает свою работу.

Рассмотрим методику работы загрузчика исполняемых файлов для среды MS-WINDOWS. Данная методика легально запускет исполняемые файлы для 32-разрядных операционных систем MS-WINDOWS 95 и MS-WINDOWS NT. Блок-схема программы LOADER в среде MS-WINDOWS приведена на рисунке 2.

В основе данной методики лежит преобразование исполняемого файла в соответствии с выбранным алгоритмом преобразования.

Перед запуском любой исполняемой программы, на которой установлена защита от нелегального запуска, происходит запуск программы загрузчика, проверяющей полномочия пользователя, производившего запуск программы. В состав проверки полномочий входит:

1) проверка пароля, введенного пользователем;

2) проверка текущей конфигурации машины и операционной системы (защита от нелегального копирования программ).

После того как пользователь произвел запуск исполняемого файла, на котором установлена защита, программа загрузчика запрашивает пароль у пользователя. После ввода пароля производится восстановление конфиденциальных параметров. Из восстановленных конфиденциальных параметров производится проверка правильности введенного пароля и полномочий пользователя. Если результат проверки отрицателен, программа автоматически завершит работу с данным пользователем.

При положительном результате проверки пароля происходит проверка конфигурации компьютера. Если данная проверка выявила

несовпадения в конфигурации машины, программа автоматически завершит работу с данным пользователем. После всех проверок программа загрузчика производит восстановление запускаемого файла в оперативную память и передает ей управление, а программа загрузчика выгружается из памяти.

Заключение. Таким образом, любая программа, на которой установлена защита от нелегального запуска, хранится на диске в преобразованном виде и доступ к ней может получить практически любой пользователь компьютера. Но получить доступ к открытому исполняемому файлу возможности практически нет, так как восстановление происходит прямо в память только на момент выполнения программы. По окончании сеанса работы программы вся занимаемая ею часть оперативной памяти очищается.

Библиографический список

1. *Мартынов А.П., Фомченко В.Н.* Криптография и электроника / Под ред. А.И. Астайкина. Саров: ФГУП «РФЯЦ-ВНИИЭФ», 2006.

2. *Джордейн Р.* Справочник программиста персональных компьютеров типа IBM PC, XT и AT. – М: «Финансы и статистика», 1992.

3. *Щербаков А.* Защита от копирования. - М: “Эдэль”, 1992.

УДК 621.317.75:519.2

А.И. Баранчиков, С.Э. Кухарев

АЛГОРИТМЫ РЕИНЖИНИРИНГА СЕКРЕТНОСТИ АТРИБУТОВ В РЕЛЯЦИОННЫХ БАЗАХ ДАННЫХ

Рассматриваются алгоритмы реинжиниринга секретности атрибутов на основе ИС с мандатным и дискреционным методами доступа. Проведен расчет оценки временной сложности данного алгоритма.

Ключевые слова*: реинжиниринг, базы данных, реляционная, проектирование, информационная система, секретность, доступ, мандатный метод, дискреционный метод.*

Введение. Целью данной статьи является разработка алгоритма, который ускорил бы процесс создания новой информационной системы на основе уже имеющейся, с учетом секретности атрибутов.

Сегодня можно говорить, что эра, когда разработчики ИС приходили в организацию и начинали проекты информатизации «с нуля», прошла. Наступает время проектов по систематической трансформации существующих ИС, или эра реинжиниринга ИС. Достижения в области IT позволили преодолеть принципиальные технические и программно-инструментальные проблемы создания и модернизации ИС, но вопрос определения семантики, а именно секретности атрибутов, остается актуальным и сейчас. Большинство этих ИС хранят в себе информацию разной степени секретности, и чтобы правильно разграничить уровни доступа во вновь проектируемой системе, необходимо определить, какие данные относятся к секретным, а какие являются общедоступными.

Подлежащие реинжинирингу части ИС могут располагаться как локально, так и быть распределены в пространстве, взаимодействуя через локальную или глобальную сеть. Исходя из этого, до применения алгоритма оценки секретности необходимо определить ключи таблиц ИС и связи между данными таблицами. Из полученных данных проектировщик может сделать определенные выводы, а именно какой метод доступа к данным использовала исходная ИС.

Реинжиниринг секретности атрибутов Будем считать, что анализ ИС на поиск ключей и связей между таблицами произведен и на его основе экспертом сделан вывод, что в ИС использовалась либо мандатная модель, либо дискреционная модель доступа к данным. Будем использовать значения в пределах от 0 до 1 для оценки выходных данных, полученных в результате работы рассматриваемых алгоритмов.

Если оценка секретности данных принадлежит 1, данные можно считать не секретными. Если оценка принадлежит отрезку от 0.5 до 1 - данные для служебного пользования. Если оценка принадлежит интервалу от 0 до 0.5 - данные секретные. Если оценка принадлежит 0, данные совершенно секретные.

Если оценка уровня доступа пользователя принадлежит 0, пользователю доступны, лишь

несекретные данные. Если оценка принадлежит отрезку от 0 до 0.5 - пользователю доступны данные для служебного пользования. Если оценка принадлежит интервалу от 0.5 до 1, доступны секретные данные. Если оценка принадлежит 1, доступны совершенно секретные данные.

Мандатный метод доступа. Мандатное управление доступом (Mandatory access control, MAC) разграничение доступа субъектов к объектам, основанное на назначении метки конфиденциальности для информации, содержащейся в объектах, и выдаче официальных разрешений субъектам на обращение к информации такого уровня конфиденциальности[2]. Для всех данных в ИС устанавливается метка одна из четырех уровней секретности: НС – несекретно, ДСП – для служебного пользования, С – секретно, СС – совершенно секретно.

Пусть дано два множества атрибутов *r(R1) и r(R2).*

R1={U1,U2...Un}, где *i* - пользователь, *а Ui* – атрибут доступа *i-го пользователя.*

R2={D1,D2...Dm}, где *j* - пользователь, *а Uj* – атрибут видимости *j-го* пользователя. Так же существует конечное множество *MAC={'НС' 'ДСП''С' 'СС'}.*

Известно условие, что данные могут быть прочитаны пользователем, если атрибут уровня доступа пользователя больше или равен атрибуту видимости данных.

Представим множество *MAC* как integer *MAC={1,2,3,4}* элементы данного множества являются атрибутами доступа и видимости.

Алгоритм реинжиниринга атрибутов секретности при мандатном управлении доступом можно описать следующим образом.

Алгоритм 1. MonitoringM

Вход: отношение R с множеством атрибутов *R1={U1,U2...Un}* и *R2={D1,D2...Dm}*

Выход: отношение R' с множеством атрибутов $R'1 = \{U'1, U'2 \ldots U'n\}$ и $R'2 = \{D'1, D'2 \ldots D'n\}$

```
BEGIN
(1)  for Ui ∈ R1 (i=1,2…n) do
(2)       for Dj ∈ R2 (j=1,2…m) do
(3)            if |Ui/Dj| < 1 then
(4)                 D'j ∈ R'2 (j=1,2 … m)= D'j ∪ 1
               else
(5)                 U'i ∈ R'1 (i=1,2 ... n) = U'i ∪ 1
          return( R'1, R'2)
END
```

Полученные на выходе алгоритма значения *R'1* при сравнении с установленным интервалом от 0 до 1 будут нести в себе информацию об уровне доступа пользователей.

Полученные на выходе алгоритма значения *R'2 1* при сравнении с установленным интервалом от 0 до 1 будут нести в себе информацию о секретности отдельных данных.

Например, есть два отношения R1(ПОЛЬЗОВАТЕЛЬ, АТРИБУТ) и R2(ДАННЫЕ, АТРИБУТ) с информацией о доступах.

Таблица 1

ПОЛЬЗОВАТЕЛЬ	АТРИБУТ
USER1	1
USER2	3
USER3	4

Таблица 2

ДАННЫЕ	АТРИБУТ
DATA1	3
DATA2	1
DATA3	2
DATA4	1
DATA5	4

Для анализа значений атрибутов методом мониторинга с использованием алгоритма строим таблицу на основе двух исходных.

Таблица 3

	D1	D2	D3	D4	D5
USER1	1/3	1/1	1/2	1/1	1/4
USER2	3/3	3/1	3/2	3/1	3/4
USER3	4/3	4/1	4/2	4/1	4/4

Анализ таблицы при помощи алгоритма MonitoringM даст следующие результаты:

USER1 = 2 из 5 = 0,4
USER2 = 4 из 5 = 0,8
USER3 = 5 из 5 = 1.

Из полученных значений можно предположить, каким уровнем доступа обладает каждый из пользователей (например, уровень USER3 – совершенно секретно).

DATA1 = 2 из 3 = 0,7
DATA2 = 3 из 3 = 1
DATA3 = 2 из 3 = 0,7
DATA4 = 3 из 3 = 1
DATA5 = 1 из 3 = 0,3.

Из полученных значений можно сделать предположение, какие данные являются секретными, а какие общедоступны (например, уровень секретности у DATA5 наиболее высокий).

Дискреционный метод доступа. Избирательное управление доступом (Discretionary access control, DAC) – управление доступом субъектов к объектам на основе списков управления доступом или матрицы доступа[2].

Пусть дано множество
R1(Di)={D1,D2...Dn}; Di={U1,U2...Um}.

Пусть дано множество

R2(Ui) = {U1, U2...Um}; Ui={D1, D2....Dn}.

Пусть заданы веса для атрибутов:

ЧТЕНИЕ = 3 (Ч), ВЫПОЛНЕНИЕ = 2 (В), ЗАПИСЬ = 1 (З)

Пусть существует идеальный пользователь, обладающий полным доступом ко всем записям в ИС,

UBEST = 6 * m.

Пусть существуют данные, доступ к которым открыт по всем критериям, для всех пользователей в ИС DBEST = 6 * n.

Алгоритмы реинжиниринга атрибутов секретности при дискреционном управлении доступом можно описать следующим образом.

Алгоритм 2. MonitoringD-1

Вход: отношение R с множеством атрибутов *R1={D1,D2...Dn}.*

Выход: отношение R’ с множеством атрибутов *R′ 1={D′1, D′2... D′n}.*

```
BEGIN
(1)     for Dk ∈ R1(k=1,2…n) do
        BEGIN
(2)       SUM= 0;
(3)         for Uij ∈ Dk (i=1,2…m) do
(4)           SUM=SUM+Ui1*Ч+Ui2*В+Ui3*З
(5)       D′k = SUM / DBEST
END
return(R′ 1)
END
```

Полученные на выходе значения *R′ 1* будут нести в себе информацию об уровне секретности данных.

Алгоритм 3. MonitoringD-2

Вход: отношение R с множеством атрибутов *R2={U1,U2...Um}.*

Выход: отношение R’ с множеством атрибутов *R′ 2={U′1, U′2... U′m}.*

```
BEGIN
(1)     for Uk ∈ R2(k=1,2…m) do
        BEGIN
(2)       SUM= 0;
(3)         for Dij ∈ Uk (i=1,2…n) do
(4)          SUM= SUM+Di1*Ч+Di2*В+Di3*З
(5)       U′k = SUM / UBEST
        END
        return (R′ 2)
END
```

Полученные на выходе значения *R′ 2* будут нести в себе информацию об уровне доступа пользователя.

Например, есть ИС с дискреционным методом доступа.

Таблица 4

		Ч	В	З
USER1	DATA1	1	0	1
USER2	DATA2	1	1	1
USER3	DATA3	0	0	0
USER1	DATA1	1	1	1
USER2	DATA2	0	0	1
USER3	DATA3	1	0	1
USER1	DATA1	0	1	0
USER2	DATA2	0	0	0
USER3	DATA3	1	1	1

Разложим ИС на множества R1 и R2.

R1 = (D1,D2,D3)

Таблица 5

	Ч	В	З
U	1	0	1
U	1	1	1
U	0	1	0

	Ч	В	З
U	1	1	1
U	0	0	1
U	0	0	0

	Ч	В	З
U	0	0	0
U	1	0	1
U	1	1	1

R2 = (U1,U2,U3)

Таблица 6

	Ч	В	З
D	1	0	1
D	1	1	1
D	0	0	0

	Ч	В	З
D	1	1	1
D	0	0	1
D	1	0	1

	Ч	В	З
D	0	1	0
D	0	0	0
D	1	1	1

Анализ при использовании алгоритма MonitoringD-* даст следующие результаты:

DATA′1 = 3+3+2+2+1+1=12 /18=0.7

DATA′2 = 3+2+1+1=7/18=0.4

DATA′3 =3+3+2+1+1=10 /18=0.6.

Из полученных значений можно сделать предположение, какие данные являются секретными, а какие общедоступны (например, уровень секретности у DATA2 выше, чем у других данных).

USER′1 =3+3+2+1+1=10/18=0.6

USER′2 =3+3+2+1+1+1=11/18=0.6

USER′3 = 3+2+2+1=8/18=0.4

Из полученных значений можно предположить, каким уровнем доступа обладает каждый из пользователей (например, USER3 обладает правами лишь для служебного пользования).

Оценка временной сложности алгоритмов. Время, которое компьютер расходует на решение задачи, пропорционально числу двоичных операций. Также важную роль для оценки может играть время выполнения одной двоичной операции или время доступа к памяти на отдельно взятом ЭВМ.

Для оценки в качестве элементарной операции возьмем операцию побитового сравнения.

Естественной мерой объема входной информации для данного алгоритма будет число

элементов, подлежащих обработке, т.е. размер входного множества. В этом случае порядок времени выполнения программы O(n) будет определяться как время выполнения в наихудшем случае, как максимум времени выполнения по всем входным данным размера n.

В общем случае время выполнения конечной последовательности программных фрагментов имеет порядок фрагмента с наибольшим временем выполнения $O(\max(f(n), g(n)))$.

Чаще всего операторы присваивания имеют некоторое постоянное время выполнения, независящее от размера входных данных, имеющее порядок O(1), т.е. время, равнозначное некоторой константе.

Общее правило вычисления времени выполнения цикла заключается в суммировании времени выполнения каждой итерации цикла.

Рассмотрим сложность алгоритма MonitoringM.

Временная сложность цикла for (1) оценивается следующим образом. В цикле for сначала инициализируется переменная цикла (сложность N+1), затем в каждом проходе проверяется, что переменная не выходит за границы выполнения цикла (Сложность N+1), и переменная получает приращение (Сложность N). В наихудшем случае сложность for – N.

Временная сложность цикла for (2) оценивается следующим образом. В цикле for сначала инициализируется переменная цикла (сложность n^2+1), затем в каждом проходе проверяется, что переменная не выходит за границы выполнения цикла (Сложность n^2+1), и переменная получает приращение (Сложность n^2). В наихудшем случае сложность for – n^2.

Временная сложность операций в теле цикла (3), (4) и (5) равна N.

Так как оценка временной сложности выбирается по наихудшему случаю, то сложность данного алгоритма $O(n^2)$.

Рассмотрим сложность алгоритма MonitoringD-*.

Временная сложность циклов for (1) и (2) - N и n^2 соответственно. Временная сложность в теле цикла для операций умножения на число и операции сложения равна N. Так как оценка временной сложности выбирается по наихудшему случаю, то сложность данного алгоритма $O(n^2)$.

Заключение. В процессе решения данной задачи рассмотрена методология анализа ИС, разработаны алгоритмы реинжиниринга секретности атрибутов для двух основных методов доступов к данным, проведена оценка временной сложности разработанных алгоритмов. Применение данных алгоритмов уменьшает время исследования структуры БД и сокращает время подготовительных этапов в процессе реинжиниринга.

Библиографический список

1. *Мейер Д.* Теория реляционных баз данных. – М.: Мир, 1987. – 608с.

2. *Соколов А.В., Шаньгин В.Ф.* Защита информации в распределенных корпоративных сетях и системах. - М.: ДМК Пресс, 2002. - 656 с.

3. *Мальцев Ю., Петров Е.* Введение в дискретную математику. Барнаул: Изд-во Алтайского государственного университета, 1997.

4. *Коблиц Н.* Курс теории чисел и криптографии.- М.: Научное изд-во ТВП, 2001.

УДК 519.1/519.2

Г.С. Орлов

ИЗМЕРИМОЕ ПРОСТРАНСТВО ОБЛАСТИ ЛОКАЛИЗАЦИИ ВЕРШИН ОБОБЩЁННОГО ДИНАМИЧЕСКОГО ВЕРОЯТНОСТНОГО НАГРУЖЕННОГО ГРАФА, ДОПУСКАЮЩЕЕ РАЗБИЕНИЕ НА ЗОНЫ ОТВЕТСТВЕННОСТИ ВЕРШИН

На основе введённого ранее автором понятия обобщённого динамического вероятностного нагруженного графа строится измеримое пространство области локализации его вершин для случая, когда эта область допускает разбиение на зоны ответственности вершин графа. Определяется функция множеств, задаётся мера, вводится соответствующий интеграл по мере, анализируются его основные свойства.

Ключевые слова: *обобщённый динамический вероятностный нагруженный граф, измеримое пространство, мера, интеграл по мере.*

Введение. Данная работа продолжает исследование свойств нового объекта (см. [1], [2], [3]). ***Целями статьи*** являются (**1**) введение нового понятия *области локализации вершин графа, допускающей разбиение на зоны ответственности вершин*; (**2**) построение *измеримого пространства области локализации вершин* обобщённого динамического вероятностного нагруженного графа и анализ его свойств.

Постановка задачи. Пусть $G=(X_G,P_G)$ - некоторый пространственно нестационарный пространственно-временной обобщённый динамический вероятностный полностью нагруженный граф (ПН П-В ОДВПНГ) [1, 2]. Предположим, что множество его вершин конечное $|X_G|=N<\infty$. Зададим некоторый временной отрезок $T=[t_{нач},t_{кон}]$, на котором в дальнейшем будем рассматривать процесс функционирования графа $G=(X_G,P_G)$. Как и раньше (*см.*[1, 2]), обозначим через $p_{\tilde{t}}(x_j)$ $(\tilde{t}\in T,\ x_j\in X_G;\ j=\overline{1,N})$ вероятность существования вершины x_j в момент $\tilde{t}$; через $p_{ij}^{(l)}(\tilde{t})$ - вероятность существования $l-$й дуги $U_{ij}^{(l)}\in P_G$ (*проведённой из вершины* x_i *графа в вершину* x_j) в момент времени $\tilde{t}$; через $C_{\tilde{t}}(x_j)$ - нагрузку вершины x_j в момент $\tilde{t}$ (*при условии, что* $p_{\tilde{t}}(x_j)\neq 0$); а через $C_{\tilde{t}}(U_{ij}^{(l)})$ - нагрузку дуги $U_{ij}^{(l)}$ (*при условии, что* $p_{ij}^{(l)}(\tilde{t})\neq 0$). Пусть $D(X_G)\subset\Re^s$ - ***область локализации*** всех вершин графа $G=(X_G,P_G)$, то есть замкнутое ограниченное подпространство некоторого $s-$мерного пространства $\Re^s$, в общем случае мощности континуум ($|D(X_G)|=\aleph$), с точками $r^{\langle k\rangle}=(r_1^{\langle k\rangle},...,r_s^{\langle k\rangle})$ которого (*и только с ними*) могут совпадать вершины графа $G=(X_G,P_G)$ в любом из допустимых временных сечений. Будем считать, что в пространстве $\Re^s$ введена некоторая система координат, определено скалярное произведение (*то есть, что* $\Re^s$ - *евклидово пространство*), а также, что пространство $\Re^s$ является метрическим пространством с метрикой $\rho(r^{\langle i\rangle},r^{\langle j\rangle})$ (*согласованной со скалярным произведением*). Требуется построить измеримое пространство области локализации вершин ПН П-В ОДВПНГ.

Приступим к решению сформулированной задачи. Прежде всего отметим хорошо известные факты (*см., например,* [6]): $\sigma-$*алгебра, порождённая замкнутыми множествами метрического пространства, совпадает с* $\sigma-$*алгеброй борелевских множеств. Причём в любом* $s-$ *мерном евклидовом пространстве за систему множеств, порождающих* $\sigma-$*алгебру борелевских множеств, можно взять систему замкнутых (а также открытых или полуоткрытых) параллелепипедов.*

Обозначим через $\Lambda(x_j)\subseteq D(X_G)$ $(j=\overline{1,N})$ (*области локализации вершин графа*) замкнутые области пространства $\Re^s$, в одну из которых может в некоторый момент времени попасть вершина x_j $(j=\overline{1,N})$ [*то есть, если* $r\in\Lambda(x_j)$ (*верхний индекс у точки* r *пространства* $\Re^s$ *будем опускать, если это не вызывает разночтений*), *то для некоторого* $\tilde{t}\in T$ *событие* $\{x_j\equiv r\mid t=\tilde{t}\}$ *(вершина* x_j *находится в точке* r *в момент времени* $\tilde{t}$ *) не является невозможным*]. В дальнейшем каждую такую область $\Lambda(x_j)$ будем называть ***зоной ответственности вершины*** x_j. Обозначим через $mes_L(A)$ $(A\subset\Re^s)$ меру Лебега для произвольного множества A метрического пространства $\Re^s$. Отметим, что множество, состоящее из одной точки пространства $\Re^s$, измеримо по Лебегу (*имеет меру нуль*). Счётное ограниченное множество точек пространства $\Re^s$ также имеет Лебегову меру нуль. Множество точек некоторого $s-$мерного гиперкуба $\Delta^{(s)}\subset\Re^s$ (*не все координаты которых рациональны*) имеет уже ненулевую меру Лебега: $mes_L(\Delta^{(s)})>0$.

Напомним (*см., например,* [4]) основные свойства меры Лебега, которыми будем пользоваться в дальнейшем.

Свойство 1. Множество A *измеримо (по Лебегу) тогда и только тогда, когда для* $\forall\, \varepsilon > 0$ *в пространстве* $\Re^s$ *найдутся такие множества* B_1, B_2, *что* $B_1 \subset A \subset B_2$ *и верно неравенство* $mes_L(B_2) - mes_L(B_1) < \varepsilon$.

Свойство 2. Если какие-то два множества измеримы по Лебегу, то измеримыми (по Лебегу) будут также и их сумма, разность и пересечение.

Свойство 3. Если ограниченное множество A *представимо в виде счётной суммы попарно непересекающихся измеримых (по Лебегу) множеств* $A = \sum_{i=1}^{\infty} B_i$, *то множество* A *измеримо (по Лебегу) и его мера* $mes_L(A) = \sum_{i=1}^{\infty} mes_L(B_i)$.

Свойство 4. Если $A_1 \supset A_2 \supset \ldots \supset A_k \supset \ldots$ *есть счётная невозрастающая последовательность измеримых множеств и* $A = \bigcap_{k=1}^{\infty} A_k$, *то* A *измеримо по Лебегу и* $mes_L(A) = \lim_{k\to\infty}[mes_L(A_k)]$

Свойство 5. Пересечение конечного или счётного числа измеримых по Лебегу множеств измеримо.

Определение 1. *Если область локализации всех вершин графа представима в виде объединения* $D(X_G) = \bigcup_{j=1}^{N} \Lambda(x_j)$ *попарно непересекающихся* $\Lambda(x_i) \bigcap \Lambda(x_j) = \varnothing \quad (i \neq j)$ *зон ответственности вершин графа* $G = (X_G, P_G)$, *то будем говорить, что область локализации вершин допускает разбиение на зоны ответственности вершин.*

Пусть в нашем случае $D(X_G) = \bigcup_{j=1}^{N} \Lambda(x_j)$ и $\Lambda(x_i) \bigcap \Lambda(x_j) = \varnothing \quad (i \neq j)$. Отметим, что мера Лебега, рассматриваемая на классе множеств $\{\Lambda(x_j)\}_{j=1}^{N}$, определяет так называемую *функцию множеств* (*см., например,* [6]). Если рассматривать всевозможные объединения множеств $\Lambda(x_j)$ из класса $\{\Lambda(x_j)\}_{j=1}^{N}$, то после добавления к ним пустого множества полученная совокупность множеств будет *алгеброй* $\alpha(\{\Lambda(x_j)\}_{j=1}^{N})$ (*алгеброй, порождённой соответствующим разбиением множества* $D(X_G) \subset \Re^s$ *на зоны ответственности вершин* [6]). Воспользуемся следующей теоремой [6]: *каков бы ни был класс множеств, существует наименьшая* $\sigma-$*алгебра, содержащая в себе этот класс множеств.* Обозначим такую $\sigma-$алгебру через $\sigma(\{\Lambda(x_j)\}_{j=1}^{N}) \overset{def}{=} \Im$. **Счётно-аддитивная функция множеств** $\mu(A) \overset{def}{=} mes_L(A)$ для любого множества $A \in \Im$ **будет** (*по определению, см.*[6]) **мерой, определённой на** $\sigma-$**алгебре множеств** $\Im$. Следовательно, **совокупность** (*тройка*) **объектов** $(D(X_G), \Im, \mu)$ **образует** так называемое **пространство с мерой**. Напомним, что любое множество $A \in \Im$ называется (*является*) $\Im-$измеримым. Отметим следующее важное свойство [6]: *любое множество* $B \in \Im$ *пространства с мерой* $(D(X_G), \Im, \mu)$ *само можно рассматривать как пространство с мерой* $(B, \Im_B, \mu_B)$, *где* $\Im_B$ *- индуцированная на множестве* B $\sigma-$*алгебра, а* $\mu_B(C) = \mu(C)$ *для* $\forall\, C \in \Im_B$.

Рассмотрим произвольную вершину $x_j \in X_G$ с соответствующей зоной ответственности $\Lambda(x_j)$. На точках $r \in \Lambda(x_j)$ зададим случайные величины (СВ) (*зависящие, вообще говоря, от временного сечения*): $\forall j = \overline{1, N}; \forall \tilde{t} \in T$:

$$\xi_{\tilde{t}}^{j}(r) \overset{def}{=} \begin{cases} 1, \textit{если } x_j \equiv r \textit{ при } t = \tilde{t}; \\ 0, \textit{в противном случае}. \end{cases}$$

Таким образом, если в некоторый момент времени $\tilde{t} \in T$ выполняется условие $\xi_{\tilde{t}}^{j}(r) = 1$, то это означает, что в данном временном сечении вершина $x_j \in X_G$ находится в точке $r \in \Lambda(x_j)$. Отметим, что из определения СВ $\xi_{\tilde{t}}^{j}(r)$ непосредственно следует, что если

$\xi^j_{\tilde{t}}(r)=1$, то $p_{\tilde{t}}(x_j)\neq 0$. Заметим, что, по сути дела, нами была определена следующая последовательность СВ:

$$\{\xi^{\,1}_{\tilde{t}}(r),\ ...,\ \xi^{\,N}_{\tilde{t}}(r)\}\ .$$

Для $\forall j=\overline{1,N}$ введённые дискретные СВ $\xi^j_{\tilde{t}}(r)$ могут принимать только два значения 0 или 1, то есть их значение чётко показывает: находится конкретная вершина x_j в точке $r\in\Lambda(x_j)$ или нет (*в момент времени* $\tilde{t}\in T$). Перейдём теперь к непрерывным СВ $\breve{\xi}^j_{\tilde{t}}(r)$, которые являются дальнейшим обобщением дискретных СВ $\xi^j_{\tilde{t}}(r)$ и множеством значений которых является весь отрезок $[0,1]$ вещественной прямой. В дальнейшем будем придерживаться следующей интерпретации. Если, например, $\breve{\xi}^j_{\tilde{t}}(r)=0{,}8$, то говорим, что в момент времени $\tilde{t}\in T$ вершина $x_j\in X_G$ может находиться в точке $r\in\Lambda(x_j)$ с вероятностью **0,8** (*при условии, что* $p_{\tilde{t}}(x_j)\neq 0$).

Предположим, что эксперимент $E[\Omega=\{\omega\}]$ состоит во вбрасывании наудачу точки (*вершины*) $x_j\in X_G$ в область $\Lambda(x_j)$, мощность которой не более чем счётная. Если точки множества $\Lambda(x_j)$ рассматривать как элементы пространства элементарных исходов $\Omega=\Lambda(x_j)$ данного эксперимента $E[\Lambda(x_j)]$, то при выполнении условия $\sum\limits_{r\in\Lambda(x_j)}\breve{\xi}^j_{\tilde{t}}(r)=1$ СВ $\breve{\xi}^j_{\tilde{t}}(r)$, определённая на $\Lambda(x_j)$, будет задавать следующую вероятность события: $P\{x_j\equiv r\,\big|\,t=\tilde{t}\}$. Рассмотрим два события, которые могут наблюдаться в рамках данного эксперимента. Событие $A_j(\tilde{t})-$ «*вершина* $x_j\in X_G$ *существует в момент времени* $\tilde{t}\in T$ »; событие $B^{\langle k\rangle}_j(\tilde{t})-$ «*в момент времени* $\tilde{t}$ *вершина* x_j *находится в точке* $r^{\langle k\rangle}\in\Lambda(x_j)$ ». Пересечение этих двух событий $A_j(\tilde{t})\bigcap B^{\langle k\rangle}_j(\tilde{t})$, очевидно, является следующим событием: «*в момент времени* $t=\tilde{t}$ *вершина* x_j *графа* G *существует и находится в точке* $r^{\langle k\rangle}$ ». Вероятность этого события равна $P(A_j\bigcap B^{\langle k\rangle}_j\,\big|\,t=\tilde{t}\,)=$

$=P(A_j\,\big|\,t=\tilde{t})\cdot P(B^{\langle k\rangle}_j\,\big|\,A_j\wedge t=\tilde{t})=$

$=p_{\tilde{t}}(x_j)\cdot\breve{\xi}^{\,j}_{\tilde{t}}(r^{\langle k\rangle})$. Если выполняется условие, что $D(X_G)$ допускает разбиение на зоны ответственности вершин, то события $\{A_j(\tilde{t})\bigcap B^{\langle k\rangle}_j(\tilde{t})\}_{j=1}^{N}$ будут независимыми в совокупности, а следовательно:

$$P\left[\bigcap_{j=1}^{N}\{A_j(\tilde{t})\bigcap B^{\langle k\rangle}_j(\tilde{t})\}\right]=$$

$$=\prod_{j=1}^{N}P\{A_j(\tilde{t})\bigcap B^{\langle k\rangle}_j(\tilde{t})\}=\prod_{j=1}^{N}p_{\tilde{t}}(x_j)\cdot\breve{\xi}^{\,j}_{\tilde{t}}(r^{\langle k\rangle})\,.$$

В том случае, когда $\left|\Lambda(x_j)\right|=\aleph$, можно задать последовательность разбиений области $\Lambda(x_j)$:

$D_1\prec\prec D_2\prec\prec\ ...\ \prec\prec D_i\prec\prec\ ...\ \prec\prec D_n$, в которой для любого индекса $k=\overline{1,(n-1)}$ разбиение D_{k+1} мельче разбиения D_k [5]. Отметим, что любое из разбиений области $\Lambda(x_j)$ также позволяет построить на $\Lambda(x_j)$ некоторую алгебру. Действительно, возьмём произвольное разбиение

$$D_k=\{D_1^{\langle k\rangle},\ D_2^{\langle k\rangle},\ ...,\ D_{m_k}^{\langle k\rangle}\}$$

с замкнутыми и ограниченными атомами $D_i^{\langle k\rangle}$. Если рассматривать всевозможные объединения множеств разбиения D_k, то после добавления к ним пустого множества полученная совокупность множеств будет алгеброй $\alpha(D_k)$ (*алгеброй, порождённой соответствующим разбиением* D_k [5]). Обозначим через $\Sigma(D_k)$ наименьшую σ- алгебру, содержащую $\alpha(D_k)$. Очевидно, что функция $\breve{\xi}^{\,j}_{\tilde{t}}(r)$, доопределённая естественным образом на

$\Sigma(D_k)$, является $\Sigma(D_k)$- измеримой. Будем считать, что на достаточно мелких разбиениях D_k (при $k >> 1$) вещественная функция $\breve{\xi}^{\,j}_{\tilde{t}}(r)$ является непрерывной. Как непрерывная функция на замкнутом и ограниченном множестве, СВ $\breve{\xi}^{\,j}_{\tilde{t}}(r)$ достигает на любом атоме $D_i^{\langle k\rangle}$ разбиения D_k своих наибольшего и наименьшего значений, обозначим их соответственно как $\max\limits_{r\in D_i^{\langle k\rangle}}[\breve{\xi}^{\,j}_{\tilde{t}}(r)]$ и $\min\limits_{r\in D_i^{\langle k\rangle}}[\breve{\xi}^{\,j}_{\tilde{t}}(r)]$. (*Отметим, что те же рассуждения проходят, если не накладывать на* $\breve{\xi}^{\,j}_{\tilde{t}}(r)$ *ограничения её непрерывности, а использовать вместо*

$$\max_{r\in D_i^{\langle k\rangle}}[\breve{\xi}^{\,j}_{\tilde{t}}(r)] \ \textit{и} \ \min_{r\in D_i^{\langle k\rangle}}[\breve{\xi}^{\,j}_{\tilde{t}}(r)]$$

соответственно

$$\sup_{r\in D_i^{\langle k\rangle}}[\breve{\xi}^{\,j}_{\tilde{t}}(r)] \ \textit{и} \ \inf_{r\in D_i^{\langle k\rangle}}[\breve{\xi}^{\,j}_{\tilde{t}}(r)]).$$

Пусть

$$\Delta(\breve{\xi}^{\,j}_{\tilde{t}};D_i^{\langle k\rangle})\overset{def}{=}\left|\max_{r\in D_i^{\langle k\rangle}}[\breve{\xi}^{\,j}_{\tilde{t}}(r)]-\min_{r\in D_i^{\langle k\rangle}}[\breve{\xi}^{\,j}_{\tilde{t}}(r)]\right|$$ - величина вариации СВ $\breve{\xi}^{\,j}_{\tilde{t}}(r)$ на атоме $D_i^{\langle k\rangle}$. Предположим, что при $n\to\infty$ для любого i - го атома n-го разбиения выполняется условие $\Delta(\breve{\xi}^{\,j}_{\tilde{t}};D_i^{\langle n\rangle})\to 0$. Зафиксируем некоторое малое $\varepsilon>0$ и построим такое разбиение D_k, у которого для любого атома $D_i^{\langle k\rangle}$ выполняется условие $\Delta(\breve{\xi}^{\,j}_{\tilde{t}};D_i^{\langle k\rangle})<\varepsilon$. Определим на атомах $D_i^{\langle k\rangle}$ полученного разбиения D_k простую функцию $\zeta^{\,j}_{\tilde{t}}(D_i^{\langle k\rangle})$ (*см., например,* [6]) следующим соотношением: $\zeta^{\,j}_{\tilde{t}}(D_i^{\langle k\rangle})\overset{def}{=}\frac{1}{2}\cdot\left(\max\limits_{r\in D_i^{\langle k\rangle}}[\breve{\xi}^{\,j}_{\tilde{t}}(r)]+\min\limits_{r\in D_i^{\langle k\rangle}}[\breve{\xi}^{\,j}_{\tilde{t}}(r)]\right)$ (*или соответственно*

$$\zeta^{\,j}_{\tilde{t}}(D_i^{\langle k\rangle})\overset{def}{=}\frac{1}{2}\cdot\left(\sup_{r\in D_i^{\langle k\rangle}}[\breve{\xi}^{\,j}_{\tilde{t}}(r)]+\inf_{r\in D_i^{\langle k\rangle}}[\breve{\xi}^{\,j}_{\tilde{t}}(r)]\right)).$$

Ясно, что $\zeta^{\,j}_{\tilde{t}}(D_i^{\langle k\rangle})$ также является $\Sigma(D_k)$- измеримой (*сумма двух* $\Sigma(D_k)$*-измеримых функций также* $\Sigma(D_k)$*- измерима, см., например,* [6]). Пусть $\{c_1, c_2, ..., c_{m_k}\}$ - множество значений простой функции $\zeta^{\,j}_{\tilde{t}}(D_i^{\langle k\rangle})$ в случае разбиения D_k. В соответствии с определением (*см.*[6]) интегралом от $\zeta^{\,j}_{\tilde{t}}(D_j^{\langle k\rangle})$ по множеству $\Lambda(x_j)$ назовём величину

$$\int_{\Lambda(x_j)}\zeta^{\,j}_{\tilde{t}}\left(D_i^{\langle k\rangle}\right)\mu(dr)\overset{def}{=}$$

$$\overset{def}{=}\sum_{i=1}^{m_k}c_i\cdot\mu\{r\in\Lambda(x_j):\zeta^{\,j}_{\tilde{t}}(D_i^{\langle k\rangle})=c_i\}=$$

$$=\sum_{i=1}^{m_k}c_i\cdot mes_L(D_i^{\langle k\rangle}). \qquad (1)$$

Заметим, что для произвольного множества $M\in\Sigma(D_k)$ имеем

$$\int_M\zeta^{\,j}_{\tilde{t}}(D_i^{\langle k\rangle})\mu(dr)=\int_{\Lambda(x_j)}\zeta^{\,j}_{\tilde{t}}(D_i^{\langle k\rangle})\cdot\chi_M(r)\,\mu(dr)$$

(где $\chi_M(r)\overset{def}{=}\begin{cases}1, \textit{если } r\in M,\\ 0, \textit{если } r\notin M.\end{cases}$). Далее воспользуемся следующим фактом (*доказательство см. [6]*): *для того, чтобы некоторая функция* $f(r)$ $(r\in M, M\in\sum(D_k))$ *была* $\Sigma(D_k)$ - *измеримой, необходимо и достаточно, чтобы она была пределом сходящейся всюду на множестве* M *последовательности простых функций.* Заметим, что в данном случае имеется в виду сходимость по мере (*то есть* $f(r)\overset{def}{=}\mu\circ\lim\limits_{n\to\infty}[\varphi_n(r)]$, *если для* $\forall\varepsilon>0$ *имеет место* $\mu\{r\in M:|\varphi_n(r)-f(r)|>\varepsilon\}\underset{n\to\infty}{\to}0$). Таким образом, по построению, при $k\to\infty$ последовательность простых $\Sigma(D_k)$ -

измеримых функций $\zeta_{\tilde{t}}^{j}(D_i^{\langle k\rangle})$ сходится по мере к $\Sigma(D_k)$ – измеримой функции $\breve{\xi}_{\tilde{t}}^{j}(r)$ (*так как сходящаяся по мере последовательность функций имеет только один предел (см., например, [6])*). Заметим, что из сходимости по мере непосредственно следует (*см. [6]*), что последовательность $\zeta_{\tilde{t}}^{j}(D_i^{\langle k\rangle})$ является также и фундаментальной по мере μ. Для того чтобы можно было определить интеграл $\int_M \breve{\xi}_{\tilde{t}}^{j}(r)\mu(dr)$ от функции $\breve{\xi}_{\tilde{t}}^{j}(r)$ по произвольному множеству $M \in \Sigma(D_k)$, применим ***теорему Лебега*** *(см., например, [6]): если* $\{\varphi_n(r)\}$ *- монотонно неубывающая последовательность неотрицательных* $\Sigma(D_k)$ *- измеримых функций и* $f(r) = \mu \circ \lim_{n\to\infty}\{\varphi_n(r)\} \pmod{\mu}$, *то возможен предельный переход под знаком интеграла:* $\mu \circ \lim_{n\to\infty}\{\int_{\Lambda(x_j)} \varphi_n(r)\,\mu(dr)\} = \int_{\Lambda(x_j)} f(r)\,\mu(dr)$.

Определение 2. *Интегралом* $\int_{\Lambda(x_j)} \breve{\xi}_{\tilde{t}}^{j}(r)\,\mu(dr)$ *от функции* $\breve{\xi}_{\tilde{t}}^{j}(r)$ *по множеству* $\Lambda(x_j)$ *будем называть* $\mu \circ \lim_{k\to\infty}\{\int_{\Lambda(x_j)} \zeta_{\tilde{t}}^{j}(D_i^{\langle k\rangle})\,\mu(dr)\}$. Таким образом,

$$\int_{\Lambda(x_j)} \breve{\xi}_{\tilde{t}}^{j}(r)\,\mu(dr) \overset{def}{=} \mu \circ \lim_{k\to\infty}\{\int_{\Lambda(x_j)} \zeta_{\tilde{t}}^{j}(D_i^{\langle k\rangle})\,\mu(dr)\}.$$

Для произвольного множества $M \in \Sigma(D_k)$ определим соответствующий интеграл как величину

$$\int_M \breve{\xi}_{\tilde{t}}^{j}(r)\,\mu(dr) \overset{def}{=} \int_{\Lambda(x_j)} \breve{\xi}_{\tilde{t}}^{j}(r)\cdot\chi_M(r)\,\mu(dr).$$

Если в некотором временном сечении $\tilde{t} \in T$ выполняется условие $p_{\tilde{t}}(x_j) \neq 0$, то потребуем, чтобы $\sum_{k\in K} \breve{\xi}_{\tilde{t}}^{j}(r^{\langle k\rangle}) = 1$ (*в случае не более чем счётного множества* $\Lambda(x_j)$, *где* K – *соответствующее индексное множество*) или $\int_{\Lambda(x_j)} \breve{\xi}_{\tilde{t}}^{j}(r)dr = 1$ (*в том случае, когда множество* $\Lambda(x_j)$ *имеет мощность континуум, а* $\breve{\xi}_{\tilde{t}}^{j}(r)$ *непрерывна на* $\Lambda(x_j)$).

Пусть

$$\chi_{\Lambda(x_j)}^{\tilde{t}}(r) \overset{def}{=} \begin{cases} 1, \text{ если } r \in \Lambda(x_j) \text{ при } t = \tilde{t}; \\ 0, \text{ в противном случае.} \end{cases}$$

Тогда $\breve{\xi}_{\tilde{t}}(r) \overset{def}{=} \sum_{j=1}^{N} \breve{\xi}_{\tilde{t}}^{j}(r)\cdot\chi_{\Lambda(x_j)}^{\tilde{t}}(r)$ будет СВ, определённой на всем множестве $D(X_G) = \bigcup_{j=1}^{N} \Lambda(x_j)$. Таким образом,

$$\int_{D(X_G)} \left\{\sum_{j=1}^{N} \breve{\xi}_{\tilde{t}}^{j}(r)\cdot\chi_{\Lambda(x_j)}^{\tilde{t}}(r)\right\}\mu(dr) = \sum_{j=1}^{N} \int_{\Lambda(x_j)} \breve{\xi}_{\tilde{t}}^{j}(r)\,\mu(dr).$$

Определение 3. *Интегралом* $\int_{D(X_G)} \breve{\xi}_{\tilde{t}}(r)\,\mu(dr)$ *от СВ* $\breve{\xi}_{\tilde{t}}(r)$ *будем называть величину* $\sum_{j=1}^{N} \int_{\Lambda(x_j)} \breve{\xi}_{\tilde{t}}^{j}(r)\,\mu(dr)$.

Рассмотрим важный частный случай. Предположим, что каждая СВ $\breve{\xi}_{\tilde{t}}^{j}(r)$ равномерно распределена и непрерывна на $\Lambda(x_j)$ (*предполагается, что* $|\Lambda(x_j)| = \aleph$). Если $p_{\tilde{t}}(x_j) \neq 0$ то, как говорилось ранее, считаем что $\int_{\Lambda(x_j)} \breve{\xi}_{\tilde{t}}^{j}(r)dr = 1$. Следовательно,

$$\int_{\Lambda(x_j)} \breve{\xi}_{\tilde{t}}^{j}(r)dr = \breve{\xi}_{\tilde{t}}^{j}\cdot\int_{\Lambda(x_j)} 1\cdot dr = \breve{\xi}_{\tilde{t}}^{j}\cdot mes_L[\Lambda(x_j)] = 1,$$

откуда $\breve{\xi}_{\tilde{t}}^{j} = \left(mes_L[\Lambda(x_j)]\right)^{-1}$. Если $\breve{\xi}_{\tilde{t}}(r) \overset{def}{=} \sum_{j=1}^{N} \breve{\xi}_{\tilde{t}}^{j}(r)\cdot\chi_{\Lambda(x_j)}^{\tilde{t}}(r)$, то

$$\int_{D(X_G)} \breve{\xi}_{\tilde{t}}(r)\,\mu(dr) = \sum_{j=1}^{N} \int_{\Lambda(x_j)} \breve{\xi}_{\tilde{t}}^{j}(r)\,\mu(dr) =$$

$$= \sum_{j=1}^{N} \left\{\left(mes_L[\Lambda(x_j)]\right)^{-1}\cdot\int_{\Lambda(x_j)} \mu(dr)\right\} =$$

$= \Big|$ *в соответствии с определением, данным формулой (1)* $\Big| =$

$$= \sum_{j=1}^{N} [\,(mes_L[\Lambda(x_j)])^{-1} \cdot \mu\{r \in \Lambda(x_j)\}] = N\,.$$

Заключение. ***Основными результатами данной работы*** являются следующие: (1) в работе определено понятие области локализации вершин графа, допускающей разбиение на зоны ответственности вершин; (2) введена соответствующая мера множества; (3) построено измеримое пространство; (4) определён интеграл от случайной величины по области локализации вершин, рассмотрены его свойства.

Библиографический список

1. *Орлов Г.С.* Динамические вероятностные нагруженные графы. Определения, свойства, области применения // Вестник РГРТУ. № 1 (выпуск 31). Рязань, 2010. – С. 48 – 57.

2. *Орлов Г.С.* Случайные процессы на базе обобщённых динамических вероятностных нагруженных графов // Вестник РГРТУ. № 3 (выпуск 33). Рязань, 2010. – С. 60 - 63.

3. *Орлов Г.С.* Пространственно нестационарные пространственно–временные динамические вероятностные нагруженные графы // Вестник РГРТУ № 4 (выпуск 34). Рязань, 2010. – С. 58 – 64.

4. *Никольский С.М.* Курс математического анализа. т. II: учебник. – 3-е изд. – М.: Наука, 1983. - 448 с.

5. *Ширяев А.Н.* Вероятность. - М.: Из–во «Наука», 1989. – 638 с.

6. *Гихман И.И., Скороход А.В.* Введение в теорию случайных процессов. – М.: Наука, 1965. - 654 с.

УДК 004.056.55

Д.В. Гуров, М.А. Иванов

ДИНАМИЧЕСКАЯ РАНДОМИЗАЦИЯ СИСТЕМЫ КОМАНД МИКРОПРОЦЕССОРА ДЛЯ ЗАЩИТЫ ОТ ЭКСПЛОЙТОВ

Рассматриваются методы защиты программ, содержащих уязвимости, от эксплойтов посредством точечных изменений архитектуры микропроцессора. В качестве примера приведена модификация микропроцессора с архитектурой IA-32. Модификации касаются аппаратного шифрования системы команд, уникального для каждого выполняющегося процесса. Показана эффективность данных модификаций как в отношении защищённости вычислительной системы, так и в отношении её производительности.

Ключевые слова: *архитектура микропроцессора, система команд, уязвимости программного обеспечения, безопасность, LFSR, рандомизация.*

Введение. В настоящее время темпы развития IT технологий очень высоки. Всё больше разработчиков программного обеспечения выходят на рынок программных продуктов, предлагая всё более сложные и разнообразные программы. При этом основное внимание уделяется расширению функциональных возможностей, и делается это во многом за счёт сокращения ресурсов на поиск и устранение уязвимостей, которые могут возникать в процессе программирования. Процесс выявления и устранения уязвимостей очень долог и трудоёмок, поскольку уязвимости напрямую на ход вычислительного процесса не влияют, а проявляются только при строго избирательном воздействии на отдельные точки системы.

Для потребителей же программного обеспечения проблема наличия уязвимостей в приобретаемом и используемом продукте опасна тем, что они работают уже с исполняемым кодом, и отследить, содержит ли программа уязвимости, нет никакой возможности. Однако использование такой программы может привести к очень серьёзным последствиям – от временного отказа в обслуживании до получения злоумышленником полного контроля над системой. Именно поэтому, если уязвимость невозможно удалить из исходного кода программы, необходимо предотвратить возможность её использования злоумышленником.

Целью данной работы является разработка метода аппаратного блокирования уязвимостей,

позволяющих злоумышленнику внедрять свой код в выполняющуюся программу.

Теоретическое обоснование предлагаемого метода. Для выбора эффективных методов противодействия уязвимостям, прежде всего, проведём их классификацию. Существующие в настоящее время уязвимости программного обеспечения можно разделить на несколько классов:

1) переполнение буфера
   а) в стеке
      - реализация через вставку кода в буфер;
      - реализация через передачу управления в другую область памяти (например, в переменную окружения);
      - вызов системной функции – ret to Lib.c;
   б) в сегментах data или bss (инициализированных и неинициализированных данных соответственно);
   в) «на куче»;
2) переполнение целого;
3) уязвимость форматной строки;
4) состязание условий;
5) уязвимости обработки интерпретаторами.

Уязвимость переполнения буфера заключается в возможности перезаписи информации, размещённой после буфера, при попытке записать в буфер данные большего объёма, чем он может вместить. В зависимости от расположения буфера могут перезаписываться адрес возврата или значения регистров кадра стека (IP, CS, BP – для буфера, размещённого в стеке), значения пользовательских переменных или адреса динамических функций (для буфера, размещённого в сегментах data, bss или «на куче»). Случаи переполнения буфера в сегментах data и bss отличаются от переполнения «на куче» механизмом реализации – непосредственная перезапись в первом случае и перезапись служебной информации, используемой для работы с динамически выделяемыми блоками памяти, во втором (операции malloc и free) [1].

Уязвимость переполнения целого связана с особенностями обработки переполнений. Компилятор игнорирует такую ситуацию, в результате, прибавив к максимальному положительному числу единицу, можно получить максимальное по модулю отрицательное число, или при присваивании переменной значения большего, чем она способна вместить, старшая часть числа будет отброшена. Это особенно опасно для ситуации, когда данное число используется в качестве индекса при адресации памяти.

Уязвимость форматной строки возникает вследствие ненадлежащего использования спецификаторов в форматной строке при работе со строками. Отсутствие спецификатора при работе со строкой может позволить злоумышленнику прочитать или перезаписать произвольный байт в памяти.

Состязание условий проявляется по причине мультизадачности системы и её доступности нескольким пользователям в одно и то же время. Последовательные операции по проверке доступности ресурса и его занятию могут быть разнесены во времени из-за вызова на исполнение другого процесса, вследствие чего программа может быть «обманута», и её действия могут быть обращены во вред системе (например, перенаправление вывода программы в системный файл посредством использования ссылки). Данная уязвимость относится не столько к исходному коду, сколько к особенностям реализации мультизадачности силами операционной системы.

Наконец, особняком стоят уязвимости обработки интерпретаторами, поскольку данные обрабатываются программой высокого уровня – интерпретатором. При передаче в качестве параметра строки, содержащей команду, которая может быть интерпретирована, злоумышленник может обеспечить выполнение кода, который вовсе не подразумевался разработчиком.

Как показывает статистика [2], именно атаки, основанные на вставке кода (то есть относящиеся к первому классу), наиболее распространены в настоящее время. Поэтому, обеспечив защиту от них, можно добиться существенного повышения защищённости вычислительной системы.

Все атаки данного класса основаны на передаче управления на вредоносный участок кода. Вредоносный код внедряется в память процесса или окружения, после чего на него передаётся управление.

В настоящее время для борьбы с вредоносными вставками в готовых программах с закрытым исходным кодом могут использоваться только надстройки ядра, прерывающие программу или минимизирующие последствия такой вставки, но такие подходы существенно снижают скорость работы программы.

Такая вставка проводится на уровне исполняемого кода, поэтому совершенно необходимым условием реализации данной атаки является знание атакующим системы команд микропроцессора, с которым работает уязвимая система. Но злоумышленник не сможет внедрить свой код, если при каждом новом вызове программы один и тот же код операции будет обозначать новую операцию и по-разному

интерпретироваться микропроцессором.

Достичь такого эффекта можно путём «динамического» шифрования системы команд. Термин «динамическое» в данном случае означает замену настоящей (классической) системы команд некоторой новой, уникальной для каждого очередного программного сегмента, загружаемого в память.

Именно на этом и основан предлагаемый метод противодействия эксплойтам.

В большинстве операционных систем каждый исполняемый файл содержит сегмент кода, который при инициализации загружается процессором в выделенный сегмент памяти. Положение этого сегмента определяется значением сегментного регистра CS [3]. Для каждого выполняющегося процесса выделяется только один сегмент кода. Это даёт возможность при загрузке сегмента кода производить шифрование данного блока, используя таблицу модификации кодов операции для их замены на новое значение. Соответственно, при загрузке программы в память происходит переопределение кодов операций для каждой команды, содержащейся в сегменте кода. Расшифрование же кода операции происходит при её извлечении для непосредственного исполнения.

Для реализации работы процессора по такой схеме следует внести изменения в его структуру. В линию передачи команды от блока предвыборки команд в регистр исполняемой команды встраивается блок, выполняющий замену подменённого кода операции на его исходное значение. Поскольку шифрование (таблица замен кодов операций) уникально для каждого сегмента кода, то на входе данный блок расшифрования должен получать как зашифрованный код операции, так и значение сегментного регистра CS. На выходе блока формируется истинный (классический) код операции. На рисунке 1 приведена схема загрузки команды на исполнение.

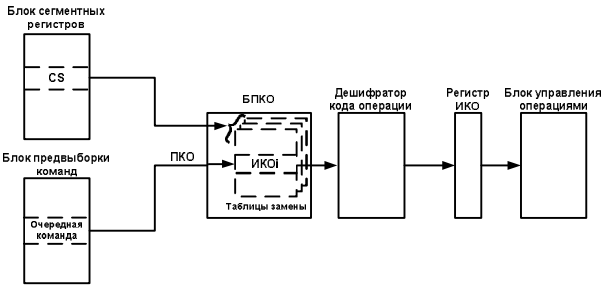

Рисунок 1 – Схема загрузки команды на исполнение

Здесь использованы следующие обозначения:

БПКО – блок подмены кода операции;

CS – сегментный регистр кода;

ПКО– подменённый код операции;

ИКО – истинный код операции.

Включение в состав процессора блока подмены требует изменения работы дешифратора кода операции. Вначале по адресу, хранящемуся в указателе команд EIP, извлекаются один, два или более байт команды (каждый последующий байт извлекается, если предыдущие оказываются префиксами, которые тоже имеют свой код для замены, чтобы скрыть себя в потоке команд). После обнаружения первого байта с собственно кодом команды анализируются 6 его первых бит, содержащих код операции. Это позволяет определить формат пост-байта, который, в свою очередь, позволяет определить количество и назначение оставшихся байт команды и тем самым вычислить адрес следующей команды (новое значение регистра EIP).

Внедрение злоумышленником своего кода в систему с предлагаемой структурой значительно усложняется вследствие того, что в архитектуре IA-32 команды имеют переменный формат, зависящий от кода операции, и, следовательно, в зашифрованном коде известно начало только первой команды (по точке входа). Таким образом, адрес каждой следующей команды можно вычислить, только восстановив код операции предыдущей команды. К тому же часть кода операции (а именно расширение кода операции для некоторых команд) может храниться во втором байте команды. Однако, не зная кода операции, невозможно сделать какие-либо предположения о том, содержит второй байт в истинной команде продолжение кода операций или какую-либо другую информацию. Следовательно, нет смысла усложнять БПКО, подменяя и второй байт для команд с распределённым кодом операции, несмотря на то, что в некоторых случаях в нём может храниться расширение кода операции.

Довольно интересен вопрос анализа условных и безусловных переходов (для нахождения начала очередных команд в других блоках при нарушении естественного порядка выборки команд), однако в данной статье этот вопрос не рассматривается.

Важно отметить, что внутри БПКО хранятся таблицы преобразований для каждого выделенного сегмента кода, которые очищаются и заполняются новыми значениями при инициализации нового процесса.

Экспериментальные исследования. Теперь подробнее остановимся на структуре и порядке работы блока подмены кода операции. В процессе зашифрования и расшифрования кода

программы путём подмены кодов операции информация пропускается через таблицу замены. В этой таблице по адресу, соответствующему коду для замены, находится новый код, на который нужно заменить поступивший. Для операции зашифрования номер строки таблицы – это классический код команды, а значение в каждой ячейке таблицы – новый код, который будет помещён в память вместо истинного. Для операции расшифрования значение и адрес меняются местами, но принцип замены тот же. Подробнее операции зашифрования и расшифрования показаны на рисунке 2.

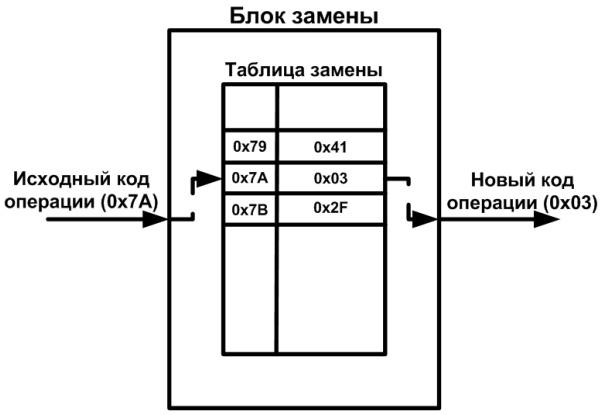

Рисунок 2 – Схема операции зашифрования и расшифрования кода операции

В работе данного блока важную роль играет формирование таблицы замен. Для этих целей предлагается использовать следующую схему (рисунок 3).

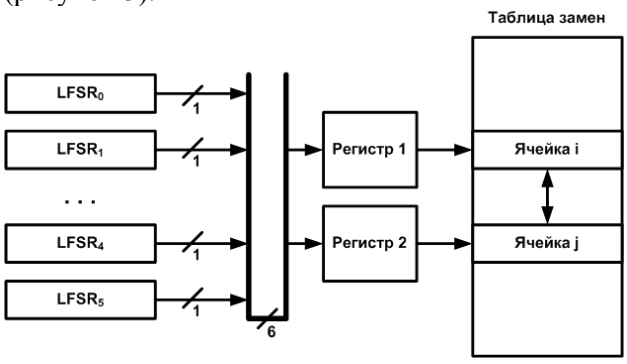

Рисунок 3 – Схема формирования таблицы замен

В основе формирования таблицы лежит использование генератора псевдослучайной последовательности на базе регистров сдвига с линейной обратной связью LFSR [4]. Использование LFSR обусловлено высокой степенью случайности, простой аппаратной реализацией и хорошим быстродействием. С каждым тактом каждый из регистров LFSR случайным образом генерирует на своём выходе значение 0 или 1, что позволяет получить на выходе всех регистров LFSR 6-разрядное случайное число, обеспечивающее адресацию таблицы объёмом до 64 ячеек. Выбор такого размера таблицы обусловлен необходимостью хранения в ней наборов 6-битных кодов, которые определяют код операции, хранящийся в первом байте команды в архитектуре микропроцессоров IA-32, а также всех возможных префиксов команд. Тут стоит напомнить, что хотя в архитектуре около 300 команд, часть из них раскрывается в постбайте, использование которого определяется по коду операции в первых 6 битах первого байта команды.

Перед началом работы (при старте системы) регистры LFSR инициализируются начальными уникальными значениями, а в каждую ячейку таблицы замен заносится её собственный адрес.

После инициализации перед началом выполнения первой программы начинается перемешивание таблицы.

Опишем одну итерацию процесса перемешивания. Она состоит из трёх тактов. Сгенерированное на первом такте работы блока LFSR-регистров 6-разрядное число записывается в Регистр 1, после чего регистры LFSR переходят в следующее состояние. Полученное в следующем такте очередное случайное число записывается в Регистр 2. Наконец, на третьем такте строки таблицы, адресуемые значениями из Регистра 1 и Регистра 2, меняются местами. На этом одна итерация заканчивается и далее всё начинается с начала: регистры LFSR сдвигаются, формируется новое значение в Регистре 1 и т.д.

Перед запуском первой программы строки таблицы замен должны быть в достаточной степени перемешаны. Проведённое в процессе исследования моделирование показало, что для того чтобы 90 % строк изменили своё местоположение по отношению к исходному, необходимо примерно 80 итераций, а для 99-процентного перемешивания требуется около 150 итераций.

Таким образом, после выполнения заданного числа итераций перемешивания получаем случайную таблицу замен, которая может использоваться при дальнейшей работе – при зашифровании и расшифровании системы команд. Генерация новой таблицы замен при загрузке очередного сегмента кода может происходить как из исходного состояния, когда в её строках записан адресный код, так и начиная с текущего состояния, поскольку в работе блока шифрования важна только сама таблица, а способ её получения не имеет значения.

Пусть в результате данного процесса получена таблица замен, фрагмент которой представлен в таблице 1.

Результат преобразования текста программы с использованием данной таблицы замен на примере программы вызова пользовательской

консоли показан в таблице 2.

Таблица 1

Исходный код операции	Код подмены
001100	010111
101100	000100
110011	111101
111010	001101
010111	011000
100010	011111
010000	001010

Таблица 2

Инструкция	Машинный код (в 16-м виде)	
	Исходный	После зашифрования
xorl eax,eax	31c0	5dc0
xorl ebx,ebx	31db	5ddb
movb 0x17,al	b017	1017
int 0x80	cd80	f580
jmp shell_str	eb1f	371f
shell_cont: popl esi	5e	62
movl esi,0x8(esi)	897608	7d7608
xorl eax,eax	31c0	5dc0
movb al,0x7(esi)	884607	7c4607
movl eax,0xc(esi)	89460c	7d460c
movb $0xb,al	b00b	100b
movl esi,%bx	89f3	7df3
leal 0x8(esi),ecx	8d4e08	014e08
leal 0xc(esi),edx	8d560c	01560c
int $0x80	cd80	f580
movl eax,ebx	89c3	7dc3
xorl eax,eax	31c0	5dc0
incl eax	40	28
int $0x80	cdx80	f580
shell_str: call shell_cont	e8dcffffff	34dcffffff

Выполнение такого машинного кода без обратного преобразования кода операции приведёт к выполнению следующих команд:

POP BP

ROL [DI+0DBh],10h

после чего работа будет прервана по ошибке в коде операции.

Из примера видно, что код программы после замены представляет собой набор случайных символов, что делает невозможным встраивание в него дополнительного кода, который выполнял бы действия, задуманные злоумышленником.

Оценка затрат на реализацию предлагаемого метода. Для анализа влияния предлагаемого метода на производительность системы оценим накладные расходы, связанные с зашифрованием/расшифрованием исходного кода. Будем считать, что система команд IA-32 представлена 64 различными кодами операций. Соответственно для каждого процесса, выполняющегося в микропроцессоре, нужно хранить таблицу преобразований объёмом в 64 строки, каждая строка которой имеет длину 12 бит (открытый/зашифрованный код операции). Если предварительно всем исходным кодам операции присвоить порядковые номера, то длина строки составит 6 бит. В этом случае, даже при хранении каждой строки в отдельном байте, получаем необходимость хранить

8x64xN = 8x64x1000 =512`000 бит

или около 64К байт информации (здесь N – расчётное количество процессов в системе, которое принято равным 1000).

Такие дополнительные затраты вполне приемлемы для современных микропроцессоров, имеющих на кристалле большие объёмы кэш-памяти.

Рассчитаем увеличение времени выполнения программы, связанное с работой БПКО. Для расшифрования кода операции требуется 1 такт (значение на выходе таблицы подмены кода операции выдаётся сразу, но необходимо дождаться следующего такта для синхронизации). Этап загрузки программы в сегмент кода требует большего времени. Здесь необходимо проанализировать каждую команду и заменить код операции в ней на зашифрованный в соответствии с таблицей замен. Расчёты, проведённые на основе учёта длительности выполнения каждой команды из системы команд и различных режимов адресации операндов [5], показали, что на этом этапе замедление составит приблизительно 4-5 % времени выполнения программы.

Выше мы показали, что формирование самой таблицы подмены занимает приблизительно 240 тактов при 90-процентном перемешивании (80 итераций по 3 такта каждая). Однако использование определённых методов оптимизации этого процесса, например, на основе распараллеливания заполнения таблицы, приведёт к существенному сокращению временных затрат на этом этапе.

Безопасность требует определённых накладных расходов, но приведённые оценочные расчеты показывают, что для предлагаемого метода они невелики.

Заключение. В данной работе рассмотрен метод противодействия атакам, использующим уязвимости, допускающие вставку вредоносного кода в программу. Предложен метод генерации таблицы замены для кодов операции в микропроцессоре на примере архитектуры IA-32.

Проведена оценка дополнительных аппаратных и временных затрат, необходимых для реализации этого метода.

Работа выполнена в рамках Федеральной целевой программы «Научные и научно-педагогические кадры инновационной России» на 2009-2013 гг.

Библиографический список

1. *Гуров Д.В., Гуров В.В., Иванов М.А., Шустова Л.И.* Технология безопасного программирования и особенности ее преподавания в вузе // – Дистанционное и виртуальное обучение. – 2010. – №9. – С.35-44.

2. *Kelly Jackson Higgins*. SANS Report: 60% Of All Attacks Hit Web Applications, Most In The U.S. [Электронный ресурс] – Режим доступа: http://www.darkreading.com/security/app-security/showArticle.jhtml?articleID=220000401

3. *Гуров В.В.* Архитектура микропроцессоров – М.: Интернет-Университет Информационных Технологий: БИНОМ. Лаборатория знаний, 2010. – 272 с.

4. Linear Feedback Shift Registers [Электронный ресурс]. – Режим доступа: http://homepage.mac.com/afj/lfsr.html

5. *Гуров В.В., Ленский О.Д., Соловьев Г.Н., Чуканов В.О.* Структура и организация вычислительного процесса в ЭВМ. – М.: МИФИ, 2003.– 108 с.

УДК 681.326

Д.Ю. Музалевский, С.И. Саитов

МОДЕЛЬ ВОЛОКОННО-ОПТИЧЕСКОГО ЛИНЕЙНОГО ТРАКТА С МУЛЬТИПЛЕКСИРОВАНИЕМ ПО ДЛИНЕ ВОЛНЫ И ВОЛОКОННО-ОПТИЧЕСКИМИ УСИЛИТЕЛЯМИ КАК ОБЪЕКТА НЕПРЕРЫВНОГО КОНТРОЛЯ

Предложена новая информационно-измерительная модель гетерогенного волоконно-оптического линейного тракта с мультиплексированием по длине волны и волоконно-оптическими усилителями как объекта непрерывного контроля в системе мониторинга мультипротокольной транспортной сети связи нового поколения.

Ключевые слова: *модель волоконно-оптического линейного тракта, интенсивность оптического сигнала, контролируемые параметры, методическая достоверность контроля, непрерывный контроль.*

Введение. Концепция *сетей связи следующего поколения* (*Next Generation Network*, *NGN*) подразумевает не только развитие собственно способов и средств обслуживания блоков данных информации, но и совершенствование сетевых обеспечивающих подсистем (синхронизации, управления, восстановления). При этом основным мероприятием обеспечения поддержания работоспособности линейного тракта (ЛТ), реализуемым при обнаружении аномалии, отказа или повреждения в оптической транспортной сети (ТС) телекоммуникационной системы (ТКС), часто является перевод транспортируемой нагрузки на резервный тракт без выяснения причин и места отказа (повреждения) [1, 2]. Это возможно при условиях:

оперативной и достоверной идентификации факта снижения качества функционирования тракта;

своевременном формировании команд на переключение нагрузки (восстановление забракованного сетевого тракта путем замены его на другой работоспособный).

Недостатками используемых сегодня систем восстановления являются:

поддержание ими какой-либо отдельной технологии переноса и избыточный расход ресурса пропускной способности линий связи, подразумевающий наличие большого числа свободных от передачи информации оптических волокон (ОВ);

использование для организации резервных каналов и трактов (по таким ОВ передаются тестовые сигналы, позволяющие осуществлять идентификацию состояния забракованных в ходе контроля волоконно-оптического линейного тракта (ВОЛТ).

Следовательно, эффективность использования ресурса пропускной способности оптической ТС ТКС может быть повышена посредством

совершенствования системы контроля ВОЛТ. Однако на этом пути существует целый комплекс нерешенных задач модельного, методического и вычислительного характера. В соответствии с изложенным вопросы моделирования ВОЛТ оптической транспортной сети *NGN* как объекта контроля являются актуальными.

Целью работы является формальное описание объекта контроля (ВОЛТ). Оно состоит в обосновании перечня *контролируемых параметров* гетерогенного ВОЛТ с мультиплексированием по длине волны (МДВ) и волоконно-оптическими усилителями (ВОУ) таким образом, чтобы:

обеспечить требования по измеримости к *контролируемым параметрам*;

сохранить полноту отражения изменений состояния объекта контроля.

Таким образом, содержанием моделирования является обоснование множества различаемых состояний оптических трактов и выбор совокупности параметров (идентификационных и прогнозирующих), которые необходимо контролировать для их различения [2, 3, 4, 5].

Современный ВОЛТ представляет собой совокупность усилительных и регенерационных секций. На рисунке 1 приведена структурная схема главного оптического тракта (ГОТ) – регенерационной секции ВОЛТ мультипротокольной транспортной сети *NGN*. Обращается внимание на то, что по спектральным каналам такого ГОТ передается гетерогенный трафик.

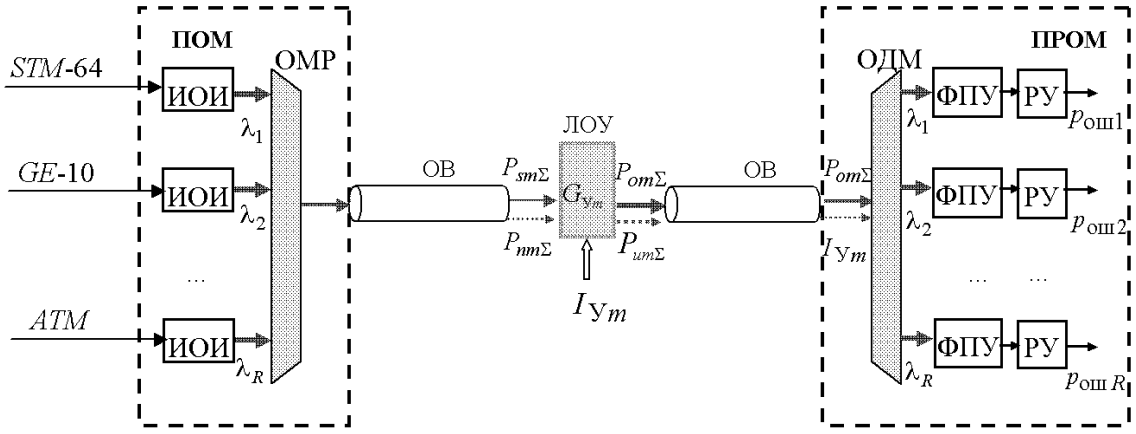

Рисунок 1 – Вариант построения мультипротокольного ГОТ с МДВ и ВОУ

Основными элементами ГОТ являются передающий оптический модуль (ПОМ), включающий в себя источники оптического излучения (ИОИ) и оптический мультиплексор (ОМ), оптическое волокно (ОВ), один или несколько линейных волоконно-оптических усилителей (ЛВОУ) и приемный оптический модуль (ПрОМ), состоящий из оптического демультиплексора (ОД) и фотоприемных устройств (ФПУ).

Так как эксплуатация ВОСП связана с получением множества экспериментальных данных (по результатам различного рода измерений и испытаний), то в ходе исследования используется информационно-измерительный подход к описанию сложных технических объектов контроля [5, 6, 7].

С внедрением новых технических решений, например систем МДВ, ВОУ и др., проявляется целый комплекс особенностей ГОТ как объекта контроля, таких как:

кратное (в десятки раз) увеличение числа управляемых объектов (и объектов контроля) на уровне среды передачи ТС;

взаимозависимость характеристик качества спектральных каналов, организованных в одном оптическом волокне;

невозможность использования рефлектометрии для организации непрерывного контроля линейных трактов ВОСП из-за однонаправленности ВОУ.

Обоснование совокупности первичных параметров, отражающих состояние главного оптического тракта. Общее состояние ВОЛТ с МДВ и ВОУ как сложного объекта оценивается исходя из назначения ТС. В традиционных оптических транспортных сетях параметры ВОЛТ с МДВ планируются (управляются) таким образом, чтобы во всех спектральных каналах обеспечивалась *равная* производительность (скорость передачи). Если хотя бы один спектральный канал является некондиционным, то весь ВОЛТ считается неработоспособным, а транспортные потоки перенаправляются в резервный тракт.

В транспортных сетях *NGN* при обслуживании трафика необходима большая вариативность состояний трактов. В этой связи в рамках проводимого исследования в качестве существенной характеристики выбрана достоверность передачи в тракте, что позволяет сформировать следующее множество различае-

мых состояний ГОТ:

исправное, при котором все спектральные каналы обеспечивают передачу потока с требуемыми скоростью и достоверностью передачи:

$$W_{\text{о}} = \{R_{\text{с}}, B_r^* \big| p_{\text{ош}\,r} \le p_{\text{ош}\,r}^*\}; \qquad (1)$$

работоспособное (промежуточное), при котором в ряде спектральных каналов допускается некоторое *ограниченное* снижение достоверности (скорости) передачи, позволяющее обеспечивать некоторые ведомственные информационные приложения:

$$W_{\text{р}} = \{s_1, B_r^* \big| p_{\text{ош}\,r} \le p_{\text{ош}\,r}^*; s_2, B_r^* \big| p_{\text{ош}\,r}^v \ge p_{\text{ош}\,r} > p_{\text{ош}\,r}^*\}$$

или

$$W_{\text{р}} = \{s_1, B_r^* \big| p_{\text{ош}\,r} \le p_{\text{ош}\,r}^*; s_2, p_{\text{ош}\,r}^* \big| B_r^v \le B_r < B_r^*\},$$
$$s_1 \ge R_{\text{пор}},\ s_1 + s_2 \ge R_{\text{кр}}; \qquad (2)$$

аварийное (неработоспособное), при котором число s_1 исправных спектральных каналов оказывается меньше порогового значения $R_{\text{пор}}$ или сумма $s_1 + s_2$ исправных и работоспособных спектральных каналов становится меньше критической величины $R_{\text{кр}}$:

$$W_{\text{а}} = \{s_1 < R_{\text{пор}} \text{ или } s_1 \ge R_{\text{пор}}, \text{ но } s_1 + s_2 < R_{\text{кр}}\}. \quad (3)$$

Нормы на величины $R_{\text{пор}}$, $R_{\text{кр}}$, $p_{\text{ош}\,r}^v$, B_r^v, s_1, s_2, устанавливаются, как правило, системой управления ТКС.

Непосредственное измерение показателей качества функционирования ВОЛТ затруднено, что обусловливает необходимость поиска параметров, наиболее полно отражающих состояние связей, организованных по спектральным каналам (СК). Проведенные исследования показали [2, 3, 4], что наилучшими характеристиками оптических сигналов и шумов с точки зрения достоверности контроля являются параметры линейного тракта $n_{\text{с}r}(t)$ и $n_{\text{ш}r}(t)$ на входе ФПУ, определяемые как среднее число фотонов, приходящихся соответственно на один сигнальный и шумовой информационный бит. Возможность измеримости параметров интенсивности оптического сигнала с требуемой точностью рассмотрена в работах [3, 4].

Обоснование способов обработки выделенной совокупности контролируемых параметров. Для получения информации о состоянии СК могут быть использованы методы приема оптических сигналов в целом [5, 8, 9] по ГОТ. Подоптимальная при приеме в целом решающая схема в процессе ее описания естественно распадается на схемы оценки параметров и собственно принятия решения.

Точность оценки параметра интенсивности на входе решающей схемы датчика контроля *r*-го СК за время наблюдения $\tau_{\text{наб}} > \tau_{\text{имп}}$ ($\tau_{\text{наб}} = N \cdot \tau_{\text{имп}}$) во многом зависит от выбора величины N – количества импульсов контрольного сигнала. На выбор величины N влияет большое количество факторов, и в конкретных условиях контроля должна быть решена задача оптимизации N по критерию несмещенности оценки

$$D_{\text{и}}(\hat{\gamma}) \approx 2\sigma_{\gamma}^2 / n_c \text{ [4].}$$

В общем случае при нахождении математического ожидания числа фотонов с заданной точностью $\delta = 0{,}01$ (δ есть не что иное как β^* – норма на величину ошибки второго рода) и надежностью $\psi = 0{,}99$ (вероятность оценивания измерения параметра с заданной точностью – имеет смысл попадания значения параметра с заданной вероятностью в доверительный интервал задаваемый $D_{\text{и}}(\hat{\gamma})$) [3, 4, 6], минимальный объем выборки, который обеспечит заданную точность, определяется выражением

$$N_{\min} = \frac{t^2 \sigma_{\text{и}}^2}{\delta^2}, \qquad (4)$$

где t – аргумент функции Лапласа, которому соответствует значение функции, равное $\psi/2$ (табулированная величина при $\psi = 0{,}99$ $t = 2{,}58$), а $\sigma_{\text{и}}^2 = D_{\text{и}}(\hat{\gamma})$.

Для дисперсии вида $D_{\text{И}}(\hat{\gamma}) \approx 2\sigma_{\gamma}^2 / n_c$, согласно формуле (4) $N_{\min} = 2064$.

Применительно к *STM*-1 и использовании, как показано на рисунке 2, свободных позиций в заголовке мультиплексной секции фрейма получаем, что для оценивания с заданной точностью необходимо обработать 2064 бит (258 байт) контрольного сигнала. Данная процедура может быть реализована за время ~ 1,88мс. Однако в реальности на точность оценки измерения будет влиять фоновый шум, учитываемый дополнительно выражением $D(\hat{\gamma}) = (2\sigma_{\gamma}^2 n_{\text{с}} + n_{\text{фш}}) \big/ (n_{\text{с}} + 1 \big/ 2\sigma_{\gamma}^2)^2$, что соответственно увеличит минимальный объем выборки [3, 6]. Только в случае $N \to \infty$ выборочные оценки являются эффективными, однако при организации трафикового контроля большое значение N противоречит требованиям оперативности контроля и эффективности использования пропускной способности СК.

Таким образом, при использовании в контрольной комбинации (КК) «0» и «1» с известным правилом чередования имеем:

$$\tau_{\text{наб}\,s} = N_{\text{КК}} \cdot \tau_{\text{имп}\,T} = N_1 \cdot \tau_{\text{имп}\,T} + N_0 \cdot \tau_{\text{имп}\,T}. \quad (5)$$

Определение длины КК, необходимой для оценивания изменений интенсивности группового ОС при воздействии s–го дестабилизирующего фактора, производится исходя из того, что закон изменения интенсивности носит экспоненциальный характер

$$\mu_{r\,s}(\gamma) = \int\limits_{t_0-\tau_{\text{наб}\,s}}^{t_0} J_r(t,\gamma)dt = \int\limits_{t_0-\tau_{\text{наб}\,s}}^{t_0} \bar{J}_{0\,r} \cdot \exp(-\gamma_s \cdot t)dt, \quad (6)$$

где $\mu_{rs}(\gamma)$ – среднее число фотонов в r–ом СК при воздействии s–го дестабилизирующего фактора, $\bar{J}_{0_r}$ – средняя интенсивность ОС.

После соответствующих преобразований и нахождения $\tau_{\text{наб}\,s}$ получаем

$$N_{\text{КК}\,s} = \frac{\ln\left(\dfrac{\mu_{r\,s}(\gamma)\cdot\gamma_s\cdot\exp(\gamma_s\cdot t_i)}{\bar{J}_{0\,r}}+1\right)}{\gamma_s\cdot\tau_{\text{имп}\,T}}. \quad (7)$$

В общем случае полученное выражение (7) отражает функциональную зависимость вида $N_{\text{КК}s} = f[\gamma_s(\beta_\text{д})]$, имеющую физический смысл количества бит контрольной комбинации, необходимых для однозначного отнесения происходящих в ГОТ процессах к конкретному виду дестабилизирующего воздействия с заданной достоверностью.

Учитывая малую длительность $\tau_{\text{имп}\,T}$ в ВОЛТ, задача оценки параметров интенсивности оптического сигнала ниже формулируется с позиций технического анализа сигналов [9]. В этом случае выбор величины N можно осуществить минимизируя ошибки аппроксимации непрерывного процесса флуктуаций энергии оптического сигнала на выходе СК. Контрольная комбинация может быть размещена, как показано на рисунке 2, в служебных полях примитивов протоколов передачи, например в поле *MSOH* кадров СЦИ.

Повышение методической достоверности контроля является одной из задач моделирования ГОТ.

Поэтому в соответствии с [4, 5] в пространстве μ_r требования к формируемой допусковой области можно записать в виде:

$$\beta_\text{д} = p(X_\text{Т} \in A_\text{р}^{(m)} / X_\text{Т} \notin S_\text{р}^{(m)}) \le \beta_\text{д}^*, \quad (8)$$

где $\beta_\text{д}^*$ – максимально допустимое значение ошибки II рода из-за замены допусковой области $S_\text{р}^{(m)}$, заданной в пространстве состояний ($K_{\text{ош}\,r}$ или $p_{\text{ош}\,r}$) областью $A_\text{р}^{(m)}$, сформированной в пространстве признаков (μ_r). В данных условиях задача оптимального преобразования $S_\text{р}^{(m)} \to A_\text{р}^{(m)}$ сводится к определению значений вероятностей ошибок I и II рода, удовлетворяющих, с одной стороны, условию (8), а с другой – обеспечивающих минимум выбранной функции потерь. Особенность решения данной задачи зависит от выбора последней.

Рисунок 2 – Вариант размещения байт контрольной комбинации в заголовке мультиплексной секции *STM-N*

Для наглядности принимается следующее представление зависимости достоверности передачи (ДП) СК от значений контролируемых параметров μ_r:

$$Q(X) = \begin{cases} 1, \mu_r \in S_\text{р}^R \\ 0, \mu_r \notin S_\text{р}^R \end{cases}. \quad (9)$$

Известно, что ошибки второго рода при решении задач на условный оптимум не приводят к ограничениям внутри системы обслуживания. Их величина ограничивается требованиями к апостериорной вероятности безотказного функционирования проконтролированного тракта [3, 4, 6]. Таким образом, при формировании функции потерь вид их учитываться не должен. То есть, если оптический тракт признан пригодным в результате контроля, то, независимо от его фактического состояния, потери от принятия такого решения внутри системы обслуживания тождественно равны 0. Если же объект ошибочно принят непригодным, потери системы обслуживания определяются затратами на восстановление ошибочно забракованного СК. Следовательно, функция потерь пропорцио-

нальна условной вероятности I рода $\alpha_д$. В рассматриваемой постановке задача оптимального преобразования $S_р^{(m)} \to A_р^{(m)}$ сводится к поиску такой области $A_{р(o)}^m$, что

$$\alpha_д(A_{р(o)}^m) = \inf_{A_{р(o)}^m \subset A_р} \alpha_д(A_р) \qquad (10)$$

и выполняется ограничение (5).

В современных ВОСП пороговые значения параметров, ограничивающие область пригодности СК по ДП, формируются, как правило, эмпирически исходя из требуемых значений $K_{ош}^*$ [4, 5].

В пространстве μ_r данный подход аналогичен нахождению экспериментальным путем величин μ_{rk}^o для каждого СК, соответствующих $K_{ош}^*$. Искомое значение μ^* будет является наибольшим из полученного множества μ_{rk}^o по итогам всех *K* обучающих примеров и всем СК:

$$\mu^* = \sup_{R,K} \mu_{rk}^o . \qquad (11)$$

Очевидно, что данный подход обеспечивает $\beta_д \to 0$. Рассмотренная процедура известна в теории контроля как аппроксимация допусковой области неправильной формы $S_р^R \equiv \{\mu_{rk}^o\}$ *R*-мерным гиперкубом $A_р^R$ с гранью, равной μ^* [4].

Геометрическое толкование этого процесса для *R* = 2 представлено на рисунке 3, *а*.

При использовании такого способа формирования допусковой области становится очевидной (в пространстве μ^*) ограниченная методическая достоверность контроля.

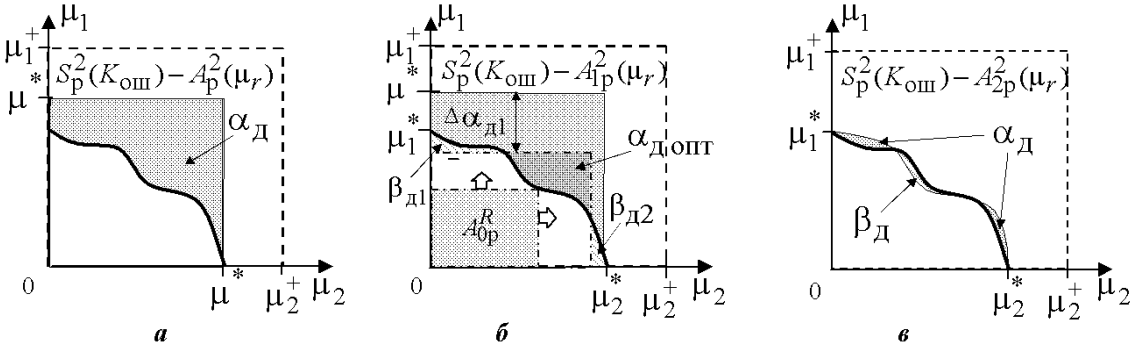

Рисунок 3 – Зависимость методической достоверности контроля от способа аппроксимации допусковой области в пространстве μ_r

Искусственное завышение порога $\mu^* > \mu_{rk}^o$ приводит к ощутимому росту $\alpha_д$ для всех СК, кроме наихудшего.

Аппроксимация допусковой области *R*-мерным параллелограммом $A_{1р}^R$ (рисунок 3, *б*) позволяет уменьшить методические погрешности введения допусков. Основным методом построения $A_{1р}^R$ в пространстве признаков μ_r, оптимальной в смысле минимума вероятности ошибки I рода является метод последовательных приближений. Начальное приближение есть вписанный в $S_р^R$ *R*-мерный параллелограмм $A_{0р}^R$, обеспечивающий $\alpha_д = 0$. При перемещении граней $A_{0р}^R$ параллельно осей координат получается совокупность $A_{iр}^R$, приближающая $(\beta_{д1} + \beta_{д2})$ к $\beta_д^*$ и увеличивающая $\alpha_д$.

При данном подходе $\beta_д$ это вероятность попадания значения μ_{rk} в область между границами $A_{iр}^R$ и $S_р^R$, при условии $\mu_{rk} \notin S_р^R$. Следовательно, используя оценки плотности вероятности $p_R(\mu_r)$, полученные в ходе испытаний ГОТ, данную задачу можно решить на основе вычисления соответствующих гиперобъемов [4]:

$$\beta_д = \frac{\int\limits_{A_{iр}^R} p_R(\mu_r)d\mu_r - \int\limits_{S_р^R} p_R(\mu_r)d\mu_r}{\int\limits_{\bar{S}_р^R} p_R(\mu_r)d\mu_r}. \qquad (12)$$

Если полученное значение $\beta_д$ удовлетворяет условию $\alpha_д(\beta_д \le \beta_д^*) \to \min$, значит соответствующая ему величина $\alpha_д$ является оптимальной. Если условие (8) не выполняется, то производится следующее приближение, заключающееся в увеличении $A_{iр}^R$, как показано на рисунке 3, *б*.

Рассмотренное справедливо только при условии постоянства значений управляемых параметров ГОТ. Изменение величин $I_{иi}$ или $I_{уi}$

потребует нового обучения и оптимизации параметров системы контроля на основе вновь полученных обучающих выборок.

Существенное повышение методической достоверности контроля может быть достигнуто в случае нелинейной аппроксимации $A_{2\text{р}}^{R}$ допусковой области $S_{\text{р}}^{R}$, как показано на рисунке 3, *в* (в приведенном примере на основе метода наискорейшего спуска [7, 8, 10]). Для используемых ранее условий задачи и на тех же статистических данных результаты моделирования на ПЭВМ (в среде *MathCad*) продемонстрировали возможность снижения $\alpha_{\text{д}r}$ до уровня $\alpha_{\text{д}r} \approx \beta_{\text{д}r}^{*} \le \beta_{\text{д}}^{*} = 0{,}01$.

Физически в рамках проводимого исследования это можно объяснить тем, что при таком решении задачи аппроксимации используется информация не только об энергетических параметрах в каждом отдельном СК, но и учитывается взаимное влияние определяющих параметров ГОТ, обусловленное наличием ВОУ, проявлением ВКР и т. д. Однако многократное осуществление процедуры нелинейной аппроксимации допусковой области для больших R и различных сочетаний $I_{\text{и}r}$ и $I_{\text{у}m}$, потребует большого числа ($K >> R$) обучающих примеров, что существенно повышает вычислительную сложность моделирования. В пространстве μ_{1r} и μ_{0r} размерность задачи аппроксимации увеличивается вдвое.

Таким образом, доказана теоретическая возможность существенного повышения достоверности контроля состояния ГОТ, значительного снижения вероятности методических ошибок контроля I рода $\alpha_{\text{д}r}$ для заданных ограничений на $\beta_{\text{д}}^{*}$. Однако большая размерность признакового пространства обусловливает высокую вычислительную сложность предлагаемых подходов к решению поставленной задачи распознавания образов и требует применения современных вычислительных методов и средств.

Информационно-измерительная модель главного оптического тракта как объекта непрерывного контроля. Модель ГОТ как объекта непрерывного контроля можно представить в виде следующей совокупности элементов.

1. Базовым объектом контроля является ГОТ. Модель ВОЛТ как объекта непрерывного контроля является объединением моделей ГОТ.

2. Основным показателем качества функционирования СК является достоверность передачи информации, выраженная величинами $K_{\text{ош}\,r}$ или, $p_{\text{ош}\,r}$, $r = 1,\ldots,R$, где R – число СК.

3. Параметрами, определяющими состояние СК, являются управляемые характеристики ГОТ $I_{\text{и}r}$ и $I_{\text{у}_m}$ ($m = 1,\ldots, M$, где M – число ВОУ) и число фотонов (фотоэлектронов) $n_{\text{с}r}$ и $n_{\text{ш}r}$, принятых ФПУ всех СК в течение бита $\tau_{\text{имп}}$.

4. Контролируемыми параметрами состояния ГОТ являются μ_r (μ_{1r} и μ_{0r}) – выборочное среднее числа фотонов, принятых за период времени $\tau_{\text{наб}\,s} = N_{\text{КК}} \cdot \tau_{\text{имп}\,T} = N_1 \cdot \tau_{\text{имп}} + N_0 \cdot \tau_{\text{имп}\,T}$, где $N_{\text{КК}} = N_1 + N_0$ – число импульсов контрольной комбинации.

5. Принятие решения о состоянии ГОТ принимается для каждого (одного) вновь измеренного вектора $\{\mu_r(t_k)\}$. Исходными данными для реализации прогнозирования являются значения управляемых характеристик ГОТ $I_{\text{и}r}$, $I_{\text{у}_m}$ и вектор $X_{\text{р}} = \{\mu_r\}$, полученный из матрицы контролируемых параметров μ_{rk}, наблюдавшихся за период ретроспекции $\tau_{\text{р}}$ во всех СК.

6. Области пригодного и непригодного состояния СК задаются значениями $\mu^{*}(\mu_1^{*}/\mu_0^{*})$, полученными в результате решения задачи оптимального формирования допусковых областей.

7. Функциональная схема ИИС, реализующей КР, представлена блок-схемой на рисунке 3.

Экономический выигрыш разработанного подхода показан ниже на основе решения задачи КР состояния ГОТ на основе общей функции потерь вида

$$C = c_{\text{к}} + \alpha \cdot c_{\text{вос}} \cdot p_{\text{бф}}, \qquad (13)$$

где $c_{\text{к}}$ – затраты на контроль (обобщенная стоимостная характеристика всех качеств реализуемого контроля, не влияющих непосредственно на ДК); α – полная условная вероятность ошибки I рода при оценке состояния отдельного СК, $p_{\text{бф}}$ – априорная вероятность безотказного функционирования тракта, $c_{\text{вос}}$ – средние затраты на восстановление ложно забракованного тракта.

В целом по ГОТ функцию потерь вида (13) для одно- и многоканального устройств контроля можно записать соответственно:

$$C_1 = Rc_{\text{к}1} + \sum_{r=1}^{R} \alpha_{1r} \cdot c_{\text{вос}r} \cdot p_{\text{бф}r}, \qquad (14)$$

$$C_{\text{мк}} = c_{\text{к мк}} + \sum_{r=1}^{R} \alpha_{\text{мк}\,r} \cdot c_{\text{вос}r} \cdot p_{\text{бф}r}. \qquad (15)$$

Анализ выражений (14) и (15) позволяет сделать вывод, что выигрыш предложенной модели определяется, во-первых, тем, что стоимость $c_{\text{к мк}}$ одного R-канального устройства контроля, как правило, меньше суммы R стоимостей $c_{\text{к1}}$ одноканальных устройств. Во-вторых, совместный анализ контролируемых параметров ГОТ позволяет оптимизировать параметры допусковых областей состояния СК, обеспечивая минимум вероятностей ошибок контроля I рода, и $\alpha_{\text{мк}\,r} < \alpha_{1r}$ для всех кроме одного СК.

Здесь следует отметить, что из-за высокой размерности сформулированной задачи контроля возникает необходимость использования средств параллельной обработки информации [6, 8, 10, 11] для исследования актуального и определения вероятного будущего состояния ГОТ.

Информационно-измерительная модель (ИИМ) ГОТ как объекта непрерывного контроля синтезируется на базе идеализированной модели в классе настраиваемых в ходе супервизорного обучения моделей.

В этом случае используется оценка

$$\xi_1 = \sum_{X \in X_a} \left| F_M(X) - Y_a \right|, \qquad (16)$$

где суммирование по X проводится по некоторому конечному набору параметров X_a, называемому обучающим множеством, для которого Y_a известны.

Норма ξ_1 - *ошибка обучения* ИИС. Для случая точных измерений однозначность системной функции F для достаточно широкого класса F_M моделей гарантирует возможность достижения произвольно малого значения ошибки обучения.

Неизвестная ошибка ξ_2, допускаемая моделью F_M на данных, не использовавшихся ранее при обучении, называется *ошибкой обобщения* модели.

Поскольку истинное значение ошибки обобщения ξ_2 не доступно, в практике используется ее оценка. Для ее получения анализируется часть примеров X_b из имеющейся базы данных, для которых известны отклики системы Y_b, но которые не использовались при обучении

$$X_b \in C_b = \{ X_b, Y_b \}, \; X_b \cap X_a = \varnothing.$$

$$\xi_2 = \sum_{X \in X_b} \left| F_M(X) - Y_b \right|. \qquad (17)$$

Выборка $C_b = \{ X_b, Y_b \}$ называется тестовой (верификационной) выборкой, а оценка ошибки обобщения является принципиальным моментом при построении информационно-измерительной модели ГОТ как объекта непрерывного контроля.

Выводы. В работе представлены основные научные результаты, полученные в ходе разработки модели ЛТ ВОСП с МДВ и ВОУ транспортных сетей связи как объекта непрерывного контроля.

Дальнейшие исследования будут направлены на подтверждение состоятельности представленных выше теоретических и практических результатов. В этой связи предполагается исследовать разработанную модель на адекватность, чувствительность и устойчивость.

Библиографический список

1. Концептуальные положения по построению мультисервисных сетей на ВСС России 2001 г. Версия 4. –М.: Министерство Российской Федерации по связи и информатизации. – 2001. – 34 с.

2. *Иванов А.Б.* Контроль соответствия в телекоммуникациях и связи. Часть I – М.: Компания Сайрус Системс, 2001. – 375 с.

3. *Советов Б.Я., Яковлев С.А.* Моделирование систем: учеб. для вузов. – М.: Высшая школа, 2001. – 343 с.

4. *Музалевский Д.Ю.* Исследование измеримости параметров интенсивности оптического сигнала / Д.Ю. Музалевский // Вестник Рязанского государственного радиотехнического университета. – 2009. – № 2. – С. 79–83.

5. *Кудрицкий В.Д.* Прогнозирующий контроль радиоэлектронных устройств. – Киев: Техника, 1982. – 168 с.

6. *Иванов А.Б.* Волоконная оптика: компоненты, системы передачи, измерения. – М.: Компания Сайрус Системс, 1999. – 671 с.

7. *Гмурман В.Е.* Теория вероятностей и математическая статистика: учеб. пособие. – М.: Высшая школа, 1999. – 479 с.

8. *Парыгин В.Н., Балакший В.И.* Оптическая обработка информации. – М.: Изд. МГУ, 1987, 141 с.

9. *Коржик В.И.* Расчет помехоустойчивости систем передачи дискретных сообщений: справочник. – М.: Радио и связь, 1981. – 232 с.

10. *Гальярди Р.М., Карп Ш.* Оптическая связь. – М.: Связь, 1978. – 424 с.

11. *Скляров О.К.* Волоконно-оптические сети и системы связи / О.К. Скляров. – М.: СОЛОН-пресс, 2004. – 255 с.