Тестирование и мониторинг в распределенных автоматизированных системах технологического управления

А. А. Кузнецов, С. П. Ковалёв
Новосибирский государственный университет, Россия
e-mail: localstorm@gmail.com

Рассмотрена проблема построения системы управления отказами в АСТУ большого масштаба. Дается сравнительный анализ существующих решений задач тестирования и мониторинга программных компонентов, предлагаются технические решения для построения промышленной системы управления отказами.

Ключевые слова: управление отказами, тестирование, мониторинг, распределенные крупномасштабные АСТУ, журналирование.

Введение

В традиционном подходе к разработке программного обеспечения считается, что задачи автоматического тестирования и мониторинга ПО не пересекаются. Но в случае, если речь идет о разработке распределенных приложений, такой подход не целесообразен. Другими словами, тестирование и мониторинг ПО следует выполнять одновременно в рамках задачи управления отказами.

Определимся с терминами, широко используемыми в данной работе.

Тестирование — это процесс анализа ПО, направленный на выявление различий между его реально существующими и требуемыми свойствами (дефект) и на оценку свойств ПО [1].

Мониторинг — это специально организованное, систематическое наблюдение за состоянием объектов, процессов с целью их оценки, контроля или прогноза.

Под *управлением отказами* в данной работе мы будем понимать определение, регистрацию, уведомление персонала и (если это возможно) автоматическое устранение неисправности, с тем чтобы позволить системе выполнять функции по ее прямому назначению. Несмотря на то, что данное определение часто применяется в области сетевых технологий, оно также применимо и в отношении распределенного ПО.

Попытки создания инструментария, пригодного для решения задач управления отказами сложных программных систем, неоднократно предпринимались ранее [2, 3]. Данные подходы безусловно интересны с научной точки зрения, но их реализации не лишены недостатков, делающих их малопригодными для широкого промышленного использования. В частности, эти подходы ориентированы на работу с программными продуктами определенных поставщиков и не являются универсальными.

Основные подходы к решению задачи управления отказами в случае распределенных программных систем — это автоматизированное тестирование и мониторинг. Нами

было показано, что совместное использование практики тестирования и мониторинга дает положительных эффект на нескольких этапах жизненного цикла системы: разработки, развертывания и промышленной эксплуатации автоматизированной системы технологического управления (АСТУ).

В данной работе рассматривается также ряд технических решений, использованных при разработке системы мониторинга промышленного уровня готовности. В частности, применение полуструктурированных документов в формате XML для решения проблемы хранения, обеспечения прозрачного доступа и визуализации сведений произвольной структуры, сохраняемых в технологическом журнале системы.

Изложенные решения были апробированы при реализации АСТУ "Интеграционная платформа учета и управления энергообеспечением" масштаба РФ, для чего была поставлена задача управления отказами. Однако не существует ограничений для применения аналогичных решений в других распределенных системах технологического управления, что позволяет говорить о возможности разработки системы управления отказами, пригодной для повторного использования. В данной работе будут также рассмотрены ограничения, при соблюдении которых такое повторное использование будет возможно.

1. Управление отказами распределенных систем

При разработке ПО неизбежно возникает задача управления отказами, отвечающего специфике разрабатываемой системы.

В случае, если мы имеем дело с нераспределенной программной системой, задача управления отказами часто сужается до корректной обработки исключительной ситуации, выдачи соответствующего сообщения об ошибке пользователю и/или занесения сообщения в журнал. После чего функционирование программной системы либо продолжается, либо (в случае серьезного сбоя) прекращается. Тестирование же программных компонентов может проводиться без учета их окружения и без надлежащей верификации заданных параметров конфигурации системы.

В случае распределенной системы подобный подход к управлению отказами не приемлем. Во-первых, в распределенных программных комплексах значительная часть сбоев бывает обусловлена ошибками развертывания или конфигурирования. Во-вторых, система сбора данных о произошедших исключительных ситуациях и отказах должна также обладать свойствами распределенной системы. Должен существовать единый интерфейс представления информации о возникших проблемах. Наконец, нужно реализовать возможности оповещения персонала, ответственного за надлежащее функционирование распределенной системы, о случаях сбоев, потере связи с удаленными компонентами системы и т. д.

Нами был проведен сравнительный анализ функциональности систем мониторинга и тестирования приложений. Требования к функциональности разделены на общие требования (относящиеся как к тестированию, так и к мониторингу), требования к системам тестирования и требования к системам мониторинга приложений.

В данном сравнительном анализе были рассмотрены четыре системы: Apache Continuum, JUnit/JUnit EE [4], IBM Autonomic Computing Toolkit (далее ACTK) и MS AsmL. Система IBM ACTK состоит из множества компонентов, однако в открытом доступе имеется только Log & Trace Analyzer. Другие компоненты IBM ACTK были описаны в публикации [2].

Обоснование выбора данных систем состоит в следующем.

IBM ACTK — наиболее известная система мониторинга и анализа функционирования распределенных систем, предназначенная для разработки средств автоматизации администрирования крупных программно-аппаратных комплексов.

MS AsmL — специфическая разработка корпорации Microsoft в области тестирования программных компонентов, отличающаяся от типичных способов автоматического тестирования (Unit-тестирование) возможностью построения модели исполняемой программы.

JUnit/JUnit EE — де-факто индустриальный стандарт в области автоматического модульного тестирования [5] для платформы Java.

Apache Continuum — система обеспечения постоянной готовности последних сборок и их автоматического тестирования для платформы Java.

Первые две системы обеспечивают непрерывный анализ работы готовых программных компонентов. Остальные продукты предназначены для многократной сборки и тестирования программных компонентов. Система Apache Continuum была выбрана в силу изначальной ориентации на работу с приложениями для платформы Java, открытости, абсолютного авторитета Apache Software Foundation в сообществе Java-разработчиков. В целом, выбранные системы покрывают основные потребности автоматизированного тестирования (исключая, пожалуй, нагрузочное тестирование) и в определенной мере покрывают потребности мониторинга.

Следует оговориться дополнительно, что в данном рассмотрении отсутствуют системы, предоставляющие возможности только мониторинга доступности компонентов распределенных систем. Таких систем существует много, но они предназначены для использования администраторами, а не разработчиками.

2. Требования к системам тестирования

В табл. 1 приведены результаты сравнительного анализа четырех выбранных систем на предмет удовлетворения ими потребностей автоматизированного тестирования.

Таблица 1. Требования к системам тестирования

Требование	IBM ACTK	MS AsmL	Apache Continuum	JUnit(-EE)
Исполнение тестового сценария	—	+	+	+
Интеграция со встроенной системой безопасности	—	—	+	+
Тестирование окружения и конфигурации	—	—	—	+
Инициализация окружения системы	—	—	+	—
Сбор информации о результатах тестирования различных компонентов	—	—	+	+
Поиск и фильтрация результатов тестирования	—	—	+	—

Из результатов сравнения хорошо видно, что основные потребности тестирования покрываются специализированными продуктами Apache Continuum и библиотеками семейства JUnit. Однако даже успешное выполнение тестовых сценариев на этапе разработки распределенной системы не дает достаточных оснований для того, чтобы сделать заключение о работоспособности системы в реальном окружении. Вот почему целесообразно интегрировать систему поддержки автоматизированного тестирования в разрабатываемую распределенную систему. Это позволяет проводить тестирование на этапе развертывания и, возможно, на этапе промышленной эксплуатации системы.

3. Требования к системам мониторинга

В табл. 2 приведены результаты сравнительного анализа четырех выбранных систем на предмет удовлетворения ими потребностей мониторинга, в том числе распределенных приложений.

Из приведенных результатов сравнения становится ясно, что требования мониторинга покрываются, хотя и частично, только одной из рассмотренных систем — IBM ACTK. Однако набор доступных компонентов ACTK на сегодня ограничен только утилитой для анализа файлов-журналов. Остальные требования, как следует из описаний и обзоров ACTK, реализованы в компонентах, которые либо уже не сопровождаются, либо недоступны для изучения.

Следует отметить, что возможность построения модели объекта управления очень важна в случае, если мы имеем дело с распределенными системами, так как чем больше масштаб системы, тем более значим административный фактор распределенности (отдельные части системы обслуживаются различными организациями или подразделениями одной организации). Отсутствие модели объекта управления не позволяет, например, определять политику доступа к данным системы мониторинга. Возможность построения модели реализована и доступна только в MS AsmL, однако остальные ха-

Таблица 2. Требования к системам мониторинга

Требование	IBM ACTK	MS AsmL	Apache Continuum	JUnit(-EE)
Мониторинг нескольких распределенных компонентов	+	—	—	—
Единый формат событий	+	—	—	—
Поиск и фильтрация событий	+	—	—	—
Разделение на два режима функционирования системы: тестовый и реальный	+	—	—	—
Редактирование уровней журналирования событий	+	—	—	—
Возможность “обучения”	+	—	—	—
Автоматическое реагирование на сбои	+	—	—	—
Построение модели объекта мониторинга и тестирования	+	+	—	—

рактеристики AsmL не позволяют рассчитывать на возможность промышленной эксплуатации разработки корпорации Microsoft.

4. Общие требования

В табл. 3 приведены результаты сравнительного анализа четырех выбранных систем на предмет удовлетворения ими потребностей, которые не могут быть отнесены к мониторингу или тестированию.

Следует отметить, что общие требования в значительной степени покрываются только продуктами, предназначенными для автоматизации тестирования, однако они не удовлетворяют важному требованию возможности непрерывной работы.

5. Общие выводы относительно рассмотренных решений

На сегодня не существует доступного и пригодного для широкой промышленной эксплуатации решения, которое покрывает и потребности непрерывного мониторинга распределенных систем, и потребности в тестировании как отдельных компонентов такой системы, так и корректности их конфигураций и взаимодействия на этапе запуска системы в эксплуатацию и на этапе промышленной эксплуатации.

Однако следует отметить, что разработки компании IBM, несмотря на скудность доступных компонентов и документации, реализуют ряд перспективных подходов. Наиболее интересные из них — развитие мониторинга и добавление возможностей автоматического реагирования на возникающие события. Действительно, такая возможность была бы полезна в распределенной системе. Во-первых потому, что реакция на возникающее событие сама по себе обобщает множество других важных возможностей. Оповещение ответственного персонала (основная функция систем мониторинга) — это реакция на возникающие сбои. Если в качестве объектов реакции рассматривать подсистемы разработанного ПО, то действия, выполняемые по расписанию, также могут быть реакцией на определенные события (например, на наступление новых календарных суток). В этом случае подсистема расписаний генерирует события и не зависит от того, что должно быть сделано в ответ на возникшее событие, — это придает бóльшую

Т а б л и ц а 3. Общие требования

Требование	IBM ACTK	MS AsmL	Apache Continuum	JUnit(-EE)
Пригодность к интеграции в существующую инфраструктуру	+	—	+	+
Автоматическая классификация данных (штатные, нештатные)	+	+	+	+
Оповещение о получении нештатных данных	+	—	+	—
Возможность работы по расписанию	—	—	+	—
Возможность непрерывной работы	+	+	—	—

гибкость при конфигурировании такой системы, а подсистема мониторинга становится важной составляющей распределенной системы.

Серьезной задачей представляется интегрирование хорошо зарекомендовавших себя инструментов автоматизированного тестирования в разрабатываемую крупномасштабную систему учета и управления энергообеспечением [6], а также дополнение полученного результата с помощью реализации собственной системы мониторинга состояния с учетом подходов, выработанных компанией IBM. В дальнейшем предполагается выделение подсистемы мониторинга состояния и средств автоматизации тестирования в отдельный продукт, пригодный для использования в других крупномасштабных системах технологического управления.

6. Реализация тестирования и мониторинга в АСТУ

Допустим, что для АСТУ выполнены следующие предположения:

1) система использует модель объекта управления, которая включает список ресурсов и их текущую конфигурацию;

2) модель объекта управления включает реестр типов событий, регистрируемых системой;

3) система предоставляет программный интерфейс для получения списка названий секций таблиц ЕТЖ по диапазону времени фиксации события. То есть существует способ отсечения секций, заведомо не содержащих требуемых данных.

Следует заметить, что для большинства (если не для любой) АСТУ данный перечень требований выполнен, а это значит, что повторное использование разрабатываемых нами программных компонентов будет возможно в достаточно широком классе АСТУ.

Подсистему мониторинга состояния можно разделить на две составные части:

1) OLTP-слой для обработки поступающих данных о событиях в средствах измерения;

2) модуль анализа событий и автоматического реагирования в случае, если событие удовлетворяет определенным критериям, которые задаются либо на этапе ввода системы в эксплуатацию, либо ответственным персоналом непосредственно в процессе эксплуатации.

7. Технические аспекты реализации подсистемы мониторинга

Основой подсистемы мониторинга считается хранилище записей о событиях, произошедших в распределенной системе, которое мы будем называть единым технологическим журналом (далее — ЕТЖ).

В зависимости от контекста будем говорить о ЕТЖ либо как о хранилище, либо как об интерфейсе доступа к хранилищу данных. Важнейшие требования к реализации ЕТЖ следующие:

1) новая запись должна добавляться за время, не большее заранее заданного, вне зависимости от количества записей, уже содержащихся в ЕТЖ;

2) ЕТЖ должен позволять произвести архивирование и удаление данных, в которых больше нет необходимости (устаревшие данные);

3) ЕТЖ должен позволять осуществлять выборки данных, удовлетворяющих заранее заданному набору критериев;

4) при добавлении новых записей в ЕТЖ подсистема реагирования должна быть оповещена о новом событии, причем такое оповещение не должно приводить к блокировке ЕТЖ до тех пор, пока оповещение не будет соответствующим образом принято подсистемой реагирования (асинхронное взаимодействие);

5) запись ЕТЖ должна соответствовать не только тому ресурсу энергосистемы, в котором произошло событие, но и его конфигурации на момент фиксации события. Иными словами, должна поддерживаться версионность конфигурации энергосистемы для последующего анализа событий;

6) задачи энергоучета требуют от ЕТЖ обеспечения высокой производительности при выборке записи о событии, имеющего максимальную дату фиксации при заданных значениях типа события и идентификатора ресурса энергосистемы;

7) формат записей ЕТЖ должен представлять собой фиксированный набор полей, однако нужно предусмотреть и возможность хранения, прозрачного доступа и визуализации дополнительных сведений, специфичных для записей определенного типа. Например, запись о синхронизации времени устройством сбора и передачи данных (УСПД) должна включать также сведения о предыдущем значении времени, которое было скорректировано, а запись о завершении обновления ПО УСПД должна содержать номер предыдущей и последней версии ПО.

Для того чтобы ЕТЖ удовлетворял данным требованиям, предлагаются следующие технические решения.

Так как все данные практически любой АСТУ хранятся в реляционной СУБД промышленного уровня (например, Oracle), то не существует альтернатив использованию секционированных таблиц для хранения записей о событиях. Секционирование позволяет ограничить время вставки записей в таблицу за счет того, что количество записей, уже сохраненных в таблице, не превышает определенного количества.

При использовании секционирования есть два варианта: использовать возможности промышленной СУБД (автоматическое секционирование) либо реализовать механизм секционирования самостоятельно (ручное секционирование). В случае ручного секционирования таблица — это набор обычных таблиц, первичный ключ которых уникален для всего их набора, что достигается с помощью использования одной и той же последовательности для генерации значений первичного ключа.

Автоматическое секционирование проще в применении, однако оно имеет ряд недостатков. Во-первых, особенности реализации секционирования, а также механизмы администрирования СУБД при работе с секциями специфичны для каждой реляционной СУБД. Во-вторых, автоматическое секционирование часто недоступно в бюджетных версиях промышленных СУБД. Поэтому целесообразно использовать механизм ручного секционирования таблиц. Данный механизм может быть реализован при помощи специальных PL/SQL-процедур и таблиц, хранящих списки секций и их текущее состояние. Описание реализации такого способа секционирования выходит за рамки данной статьи, отметим только, что секционирование, как правило, ведется по дате фиксации события. Такой подход обеспечивает выполнение первых двух основных требований.

Для асинхронного оповещения подсистемы реагирования могут быть применены различные подходы, основные из них — два:

1) осуществлять периодическую выборку новых записей ЕТЖ, обрабатывать выбранные события (или инициировать такую обработку другими подсистемами) и запоминать идентификатор последнего из обработанных событий. Данный подход соответствует периодической модели архитектур реального времени [7, 8];

2) использовать инфраструктуру асинхронного обмена сообщениями, такую как JMS (доступны множества реализаций JMS API [9], такие как Oracle Streams AQ). Данный подход соответствует реактивной модели архитектур реального времени [7].

Также можно комбинировать оба подхода. Рассмотрим основные достоинства и недостатки этих двух подходов. Периодическая выборка новых событий из ЕТЖ обладает следующими достоинствами:

1) данные о событиях уже сохранены в базе данных (обработка событий работает с тем же уровнем изоляции транзакций, что и все остальные действия в системе);

2) данные из системы никуда не исчезают, поэтому можно пытаться обработать события снова и снова в случае ошибок (однако на практике это достаточно сложно сделать с приемлемым качеством, например, гарантировать отсутствие повторных срабатываний и сохранение скорости обработки новых событий за счет повторной обработки большого количества старых записей, приводящих к сбоям);

3) полная независимость реализации от инфраструктурных компонентов (реализации системы обмена сообщениями), за исключением СУБД.

Периодическая выборка имеет следующие недостатки:

1) сложно балансировать нагрузку при обработке возникающих событий (выборка производится только одним потоком или сервером в кластере, иначе необходимо решать проблемы конкурентной обработки одних и тех же событий);

2) сложно обеспечить низкое время реакции без увеличения нагрузки на СУБД (частые выборки);

3) практически нет возможностей по настройке, кроме периода выборки событий.

Асинхронный обмен сообщениями обладает следующими достоинствами:

1) низкое время задержки между моментом записи события в ЕТЖ и его обработкой;

2) относительная простота балансировки нагрузки путем размещения получателей уведомлений на разных серверах кластера;

3) богатые возможности конфигурирования очередей уведомлений, возможность выбора хранилища для необработанных уведомлений (СУБД/ФС/ОЗУ...).

В качестве недостатков обмена сообщениями следует отметить следующие:

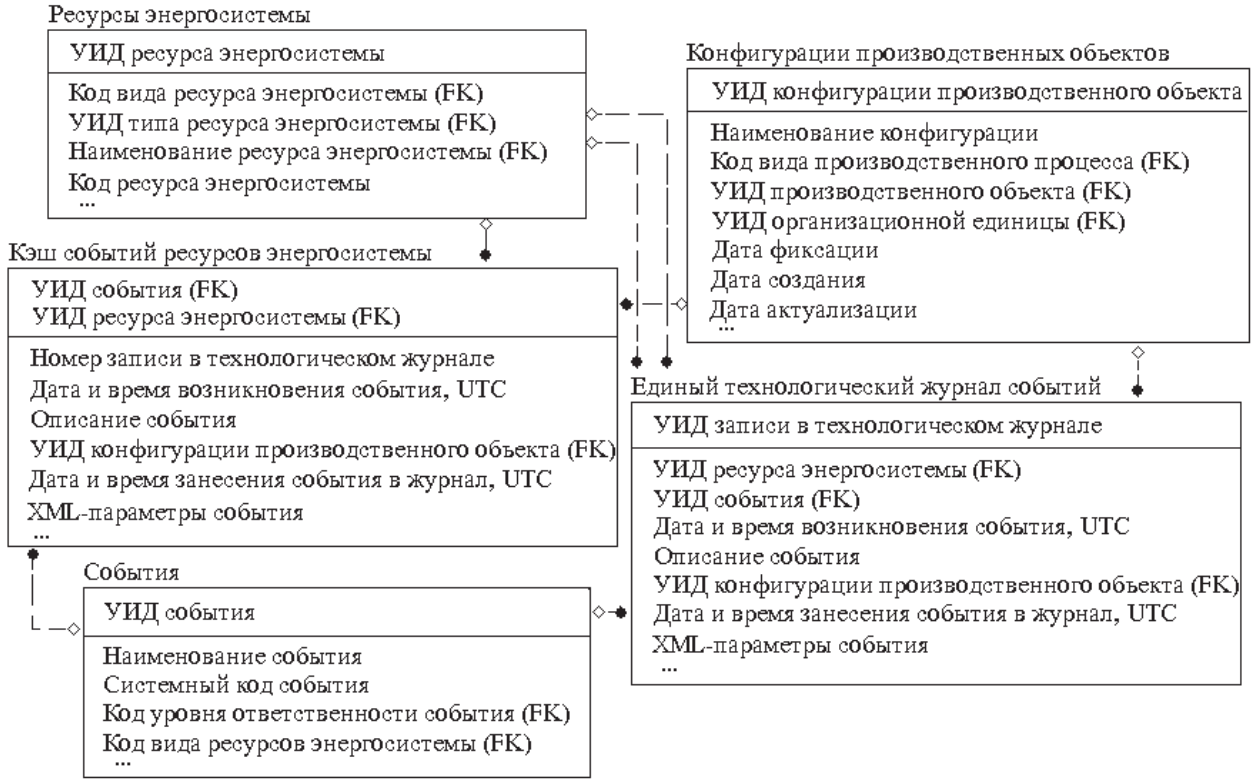
1) конечный результат (в том числе производительность) зависит от возможностей конкретного сервера приложений;

2) необработанные должным образом уведомления исчезают из очереди (возможно, после нескольких попыток обработки), и поэтому некоторые события могут так и остаться необработанными;

3) возможность хранения истории отправленных уведомлений необходимо реализовывать самостоятельно.

Учитывая перечисленные достоинства и недостатки, мы предлагаем использовать асинхронный обмен сообщениями с дополнительным ведением истории уведомлений.

Для обеспечения быстрой выборки данных о последнем зафиксированном событии для заданного типа события и ресурса энергосистемы предлагается применять специальный кэш. Кэш в данном случае представляет собой специальную секцию ЕТЖ, обновляемую триггером во время вставки записей в любую из остальных секций ЕТЖ. Первичный ключ этой секции, в отличие от остальных секций, состоит из двух полей: идентификатор ресурса и тип события, что гарантирует для данной секции наличие не более одной записи ЕТЖ для любой пары ресурс—тип события. В то же время поле



Логическая схема данных

идентификатора записи ЕТЖ не является ключевым (см. рисунок), что дает возможность архивировать и удалять секции ЕТЖ.

Конфигурация производственного объекта на момент занесения записи в ЕТЖ также определяется во время вставки с помощью триггера путем выборки актуальной конфигурации из модели объекта управления.

Для того чтобы запись ЕТЖ могла содержать дополнительные сведения произвольной структуры, целесообразно использовать полуструктурированные документы в формате XML. Данные документы сохраняются в таблицах ЕТЖ в виде BLOB и позволяют сохранять дополнительные параметры для каждой записи ЕТЖ, не создавая отдельной таблицы для событий каждого типа. Этот подход также существенно упрощает процедуру ведения секций технологического журнала, так как наличие ограничений ссылочной целостности требует, чтобы таблицы, имеющие такие ограничения, секционировались совместно, что в данном случае не оправдано.

Отметим также, что к данным, сохраненным в виде XML-документа, может быть применен механизм прозрачного доступа. Для этого существуют специализированные средства, в частности JAXB [10]. JAXB позволяет поставить в соответствие схеме документа XML класс языка Java и предоставляет достаточный инструментарий как для сохранения экземпляров данного класса в формате XML, так и для превращения XML-документов в экземпляры соответствующих классов языка Java. Данное техническое решение позволяет также использовать унифицированное решение задачи визуализации дополнительных параметров записей ЕТЖ, что становится возможным благодаря применению XSL-преобразований к сохраненным XML-документам. Для этого достаточно хранить XSL-стиль в записи справочника событий и предоставить способ доступа

к данным ЕТЖ, позволяющий в зависимости от потребностей получать либо экземпляры классов языка Java, либо исходный XML-документ.

По результатам испытаний на серверном оборудовании начального уровня, скорость вставки записей в секционированную таблицу ЕТЖ составляет от 60 до 300 записей в 1 с. Скорость существенно зависит от выбранного способа устранения дублирования записей ЕТЖ. Минимальная производительность была получена при использовании самого простого способа устранения дублирования с помощью предварительной выборки данных из ЕТЖ; максимальная производительность достигается, если дублирование записей ЕТЖ допускается.

Однако в силу того, что сбор данных о событиях, произошедших со средствами измерения, является циклическим процессом, в определенные моменты времени возможно резкое увеличение темпа регистрации событий. И, несмотря на принятые технические решения, пиковая производительность СУБД и системы в целом при добавлении новых записей в ЕТЖ остается удовлетворительной для обеспечения приема данных с большого количества средств измерения. Упрощая эту проблему производительности, предлагаем применение специального технического решения. Для целей обработки данных о событиях средств измерения создается набор из нескольких таблиц временных данных (далее “временные таблицы”), структура которых соответствует формату отчетов, содержащих сведения о событиях. Все поля этих таблиц, кроме даты вставки и идентификатора записи, строковые. Такие таблицы не содержат индексов и ограничений. Первоначальный прием отчетов о событиях сохраняется во временных таблицах без какой-либо предварительной обработки. Фоновый процесс, соответствующий каждой временной таблице, осуществляет выборку из временной таблицы, проверяет корректность полученных данных и вставляет запись в ЕТЖ даже тогда, когда данные о средствах измерения уже не поступают. Скорость вставки записей в одну временную таблицу составляет порядка 3 тыс. записей в 1 с и является приемлемой для первоначального приема данных в моменты пиковой нагрузки системы.

8. Полученные результаты

Описанные подходы были опробованы в рамках проекта “Интеграционная платформа учета и управления энергообеспечением”, далее — “Интеграционная платформа”, которая представляет собой промежуточный слой, обеспечивающий прозрачый обмен данными между техническими средствами энергоучета и системами корпоративного управления.

Технические средства отвечают за получение и регулирование “натуральных” показателей объема энергообеспечения, измеряемых в физических единицах (кВт · ч, ккал и др.). Образуется автоматизированная информационно-измерительная система (АИИС) — сложное средство измерений, выполняющее сбор и обработку данных, опираясь на физическое и метрологическое содержание процесса измерений: состав и характеристики счетчиков и датчиков расхода энергоносителей, линии передачи энергоносителей и т. д. Это содержание образует модель объекта управления, задающую правила расчета учетных показателей, потерь в энергосетях, суммарных показателей отпуска и потребления энергии. Только на уровне модели объекта управления можно распознавать расхождения план/факт энергообеспечения и выявлять их причины комплексным анализом массивов результатов измерений, событий средств измерений и объектов управления. Телемеханическими средствами (силовые выключатели с дистан-

ционным управлением, регулируемые автотрансформаторы и т. п.) достигается управление энергообеспечением.

В свою очередь, средства корпоративного управления обеспечивают функцию биллинга — превращение натурального показателя энергоснабжения в финансовый путем умножения на тариф. От них обычно требуется поддержка гибких многовариантных тарифных планов, начисление и исполнение штрафных санкций, обмен электронными документами с контрагентами и расчетно-кассовыми центрами. Корпоративное управление также включает бухгалтерский учет основных средств, задействованных в процессе энергообеспечения, организацию их технического обслуживания и ремонта и т. д. Поэтому здесь автоматизация должна быть ориентирована на поддержку делопроизводства, а не на измерение и расчет физических величин.

Глубокие функциональные различия между техническими и управленческими средствами требуют создания промежуточного слоя – интеграционной платформы. Содержание информационного обмена составляет следующая информация:

1) нормативно-справочная информация, описывающая модель объекта управления;

2) результаты измерений расхода энергоносителей;

3) события объектов и средств измерения и управления;

4) команды диспетчерского управления.

В нашем случае, события объектов и средств измерения и управления поступают на вход разрабатываемой подсистемы мониторинга состояния.

Использованные подходы позволили получить систему мониторинга состояния системы большого масштаба, обладающей высокой степенью территориальной распределенности.

В интеграционную платформу были также добавлены средства автоматизированного тестирования на основе библиотеки JUnitEE, что помогло автоматизировать тестирование на этапе разработки (для автоматизации развертывания системы и удаленного исполнения тестов системы была использована система Atlassian Bamboo, аналог Apache Continuum) и дало возможность своевременно выявлять проблемы с интеграцией различных компонентов в составе самой интеграционной платформы, разворачиваемой в тестовом окружении. Таким образом, мы получили “систему мониторинга” работоспособности исходного кода интеграционной платформы. Использование JUnitEE в составе сборки интеграционной платформы помогло также выполнить наборы тестов на этапе развертывания системы, а значит, тогда же уменьшить вероятность ошибок в реальном окружении. Открытым в данном случае остается вопрос о том, как следует разрабатывать тесты, с тем чтобы их можно было исполнять в момент запуска или во время непосредственного функционирования системы в реальном окружении, однако сейчас задача разработки таких тестов не ставится. Предполагается также, что в дальнейшем результаты самотестирования будут доступны для просмотра в едином технологическом журнале системы мониторинга.

Существует формальный способ классификации полученного результата в соответствии с моделью “автономной зрелости” автоматизированных систем [11], разработанной компанией IBM:

1) базовый уровень — системное администрирование выполняется людьми, без какой-либо автоматизации;

2) контролируемый уровень — автоматизирован сбор данных о состоянии администрируемых ресурсов;

3) уровень прогнозирования — автоматизирован анализ собранных данных;

4) адаптивный уровень — система использует результаты анализа и способна к автоматическому реагированию;

5) автономный уровень — администрирование полностью автоматизировано, система принимает решения, опираясь на определенные политики.

Применение разработанных подходов к управлению отказами позволяет приблизиться к адаптивному уровню автономности в предложенной классификации. В нашем случае набор реакций, а также условия, выполнение которых вызывает реакцию, жестко заданы (например, оповещение о критических событиях или запрос данных от УСПД, связь с которым была восстановлена).

Отметим также, что достижение пятого уровня автономной зрелости потребует значительного расширения модели объекта управления, обогащения ее данными для решения оптимизационных задач и формализации понятия политики [12] для разрабатываемой АСТУ. Выработка общего подхода к решению данной проблемы для широкого класса АСТУ выходит за рамки данной статьи.

9. Дальнейшее развитие системы

Дальнейшее развитие системы тестирования и мониторинга будет вестись в следующих направлениях:

1) обеспечение интеграции средств тестирования и мониторинга с занесением результатов тестирования в ЕТЖ;

2) добавление соответствующих реакций на возникающие события, например, автоматический запрос данных со счетчиков после восстановления связи, связь с которыми была ранее утеряна;

3) повторное использование разработанной подсистемы в других АСТУ.

Список литературы

[1] IEEE Std 829-1983 IEEE Standard For Software Test Documentation. N.Y.: IEEE, 2004.

[2] Melcher B., Mitchell B. Towards an Autonomic Framework: Self-Configuring Network Services and Developing Autonomic Applications // Intel Technology J. 2004. Vol. 8. P. 279–290.

[3] Barnett M., Schulte W. Contracts, components and their runtime verification on the .NET platform // Technical Report MSR-TR-2002-38. Redmond: Microsoft Research, 2002.

[4] Beck K. JUnit Pocket Guide. Sebastopol: O'Reilly Media, 2004.

[5] IEEE Std 1008-1987 IEEE Standard for Software Unit Testing. N.Y.: IEEE, 2004.

[6] Ковалёв С.П. Применение онтологий при разработке распределенных автоматизированных информационно-измерительных систем // Автометрия. 2008. Т. 44, № 2. С. 41–49.

[7] Berry R.F., McKenney P.E., Parr F.N. Responsive systems: An introduction // IBM Systems J. 2008. Vol. 47. P. 197.

[8] Benveniste A., Caspi P., Edwards S.A. The synchronous languages: twelve years later // Proc. IEEE. 2003. Vol. 91, N 1. P. 65.

[9] Monson-Haefel R., Chappell D.A. Java Message Service. Sebastopol: O'Reilly Media, 2000.

[10] JSR 222: Java Architecture for XML Binding (JAXB) 2.0 // http://www.jcp.org/en/jsr/detail?id=222, 2006.

[11] Worden D. Understand autonomic maturity levels // http://www.ibm.com/developerworks/autonomic/library/ac-mature.html. Armonk, 2004.

[12] Brent S., Miller A. The autonomic computing edge: The path to level 5, full autonomic maturity. Armonk: IBM Developer Works, 2005.

Поступила в редакцию 17 ноября 2008 г.,
в переработанном виде — 30 марта 2009 г.