Jet Info

ИНФОРМАЦИОННЫЙ БЮЛЛЕТЕНЬ № 8 (159)/2006

Мониторинг приложений

КОРПОРАТИВНЫЕ СИСТЕМЫ

Мониторинг приложений

Владимир Цишевский,
ведущий консультант отдела проектирования
вычислительных комплексов

СОДЕРЖАНИЕ

Первая часть предлагаемой статьи содержит обобщение шестилетнего опыта компании «Инфосистемы Джет» в сфере построения систем диагностики (мониторинга) заказных приложений. Особое внимание уделяется специфике процесса построения систем *мониторинга прикладных компонентов* в сравнении с аналогичными процессами, относящимися к *мониторингу системотехники*. Кроме того, рассматриваются различия процессов разработки и сопровождения модулей мониторинга и процессов разработки и сопровождения «традиционных» программных продуктов.

Во второй части статьи описывается архитектура промышленного программного комплекса Patrol компании BMC Software, на основе которой было реализовано значительное количество наших проектов.

Слово о Мониторинге

Прежде чем говорить о мониторинге приложений, выясним сначала, чем же так страшны эти самые приложения, что их надо как-то особенно «мониторить». Затем будет рассказано о том, как должна выглядеть «идеальная» система мониторинга и что необходимо знать, чтобы ее внедрение в организации не закончилось «как всегда».

«Маленькие» большие проблемы или Зачем нужен мониторинг?

Итак, чем опасны приложения (читай «программное обеспечение»).

К настоящему моменту около 90% бизнес-процессов коммерческих и государственных предприятий автоматизировано с использованием компьютеров и, соответственно, программного обеспечения (ПО). ИТ сегодня есть абсолютная необходимость в любом бизнесе, проблемы в ИТ способны уже повлиять не только на жизнеспособность отдельного бизнеса, но даже на национальные экономики. Сфере этой уделяется огромное внимание, отсюда разумно было бы предположить, что и программное обеспечение должно становиться с каждым днем лучше и надежнее.

В действительности это не так. ПО все более усложняется, что приводит к экспоненциальному росту количества связей и зависимостей между компонентами, и в результате надежность программ в целом снижается.

Вот печальная статистика:

- на 1000 строк кода программ приходится 100 — 150 *ошибок*[1];
- из всех проектов разработки ПО лишь 29% завершены успешно; 53% завершены с проблемами; 18% провалены[2];
- 80% приложений до внедрения *не тестируются* вообще или тестируются вручную, без использования специализированных методик и соответствующего обеспечения[3];
- *сопровождение* программных продуктов «съедает» 60-80% затрат на весь производственный цикл;
- 40% времени разработчики тратят не на разработку, а на поддержку уже «боевой» системы;

1 National Institute of Standards and Technology (NIST), 2002
2 2004 CHAOS Report, the Standish Group Gartner, Inc.
3 Researchers from Carnegie Mellon University

- 40% от времени *незапланированного простоя* приложений обусловлено ошибками в них[4].

Несмотря на это, большинство компаний рассматривают задачи диагностики как второстепенные и затратные. Но скупой платит дважды, и, как видно из приведенных далее историй, в результате даже относительно небольшие технологические проблемы могут обратиться для всего бизнеса крупной катастрофой.

2003 год. Север США и Канада. Массовые перебои в электроснабжении

Во вторник 14 августа 2003 года на северо-востоке Соединенных Штатов и в восточной части Канады произошли крупные перебои в снабжении электроэнергией. Это был наиболее крупный инцидент подобного рода за всю историю США. Пострадало около 10 миллионов жителей канадской провинции Онтарио (около трети всего населения Канады) и 40 миллионов жителей восьми американских штатов (примерно седьмая часть населения США). Принесенный ущерб оценивается примерно в шесть миллиардов долларов.

Согласно отчету американо-канадской комиссии по расследованию причин катастрофы от 19 ноября 2003 года, одной из главных ее причин стали проблемы с автоматизированной системой управления компании FirstEnergy, которая не предприняла вовремя спасательных действий, а также не оповестила о надвигающейся катастрофе другие управляющие центры. Произошло это из-за ошибки в программном обеспечении системы XA/21 производства компании General Electric Energy. Система контроля не подала необходимый сигнал об аварии, а обслуживающий персонал не обладал достаточной квалификацией, чтобы заметить неадекватное поведение системы. В течение полутора часов никто даже не догадывался, что система, призванная предупреждать об отклонениях параметров энергосети от нормы, просто не функционирует. В результате аварийная ситуация и соответствующий риск перегрузки привели к лавинообразному отключению около сотни других электростанций.

Август 2000 года. Банк Barclays (Англия). Полная остановка обслуживания

В августе 2000 года после неудачного апгрейда ПО в банке Barclays все его активные интернет-пользователи внезапно получили доступ к информации о счетах друг друга. Устранение ошибки потребовало полной 10-часовой остановки обслуживания.

Следующий пример показывает, к чему может привести недооценка влияния ошибок в программном обеспечении на бизнес в целом.

2004 год. Hewlett-Packard, неудачная миграция в SAP

Летом 2004 года в компании Hewlett-Packard стартовал проект миграции системы заказов серверов в SAP. В процессе внедрения из-за непредусмотренного заранее расхождения между теорией и практикой заказы определенных видов «застряли» между старой и новой системами. С технической точки зрения проблема не рассматривалась как критическая и была устранена за несколько недель. Однако для бизнеса последствия неожиданно оказались весьма серьезными. В течение всего лета были задержаны заказы на сумму в $120 млн, а конкуренция в торговле серверами весьма высока, и сам род бизнеса подразумевает своевременность поставок. В результате значительная часть клиентов ушла в Dell или IBM, чистый ущерб достиг $40 млн, что превышало бюджет всего проекта миграции ($30 млн)[5].

В целом, по подсчетам аналитиков, годовой ущерб от сбоев в программном обеспечении в США составил в 2004 году *$60 млрд*, что составляет 0,6% валового продукта страны.

Конечно, системы мониторинга, о которых пойдет речь ниже, не есть «серебряная пуля», делающая вычислительные комплексы абсолютно надежными и полностью свободными от ошибок. Но они могут существенно облегчить тестирование приложений еще на стадии разработки, позволят наблюдать актуальную картину «уровня здоровья» системы и выявлять корневые причины сбоев в «боевой» системе, а в конечном счете, по накопленной статистике даже оценивать заранее возможные последствия тех или иных действий, выделять «особо опасные» участки и/или вообще дадут возможность пересмотреть подход к выполнению тех или иных действий.

Например, в случае с HP, ошибка руководства компании заключалась не в допущении самих сбоев ПО (полностью избавиться от них все равно нельзя), а в непроведении анализа возможного влияния сбоев на весь бизнес — дейст-

4 Gartner Inc
5 http://123suds.blogspot.com/2004/12/when-bad-things-happen-to-good.html

вии, которое не может быть произведено силами только лишь ИТ. Прежде чем заменять «боевую» систему, надо было провести прогон на параллельном стенде или хотя бы перед переходом на новую версию сохранить действующий резервный экземпляр старой.

Теперь, обосновав актуальность темы, вернемся к вопросам диагностики ПО. Первая типичная ситуация, с которой мы сталкиваемся, когда предлагаем свои услуги, представляется примерно так: «У нас уже есть система диагностики, которая прилагается к нашей прикладной системе». Итак, мониторинг как побочный продукт производства...

Мониторинг как побочный продукт производства

Речь пойдет о средствах отладки и диагностики, поставляемых разработчиками вместе с прикладной системой. Вообще-то совсем без инструментария подобного рода невозможно ни разработать, ни эксплуатировать ни один нетривиальный программный продукт, поэтому можно считать, что инструментарий этот всегда есть при каждой программной системе.

Ради справедливости стоит отметить, что несомненным преимуществом такой ad-hoc диагностики является то, что создаются эти инструменты теми же людьми, что и сами объекты диагностики, поэтому люди эти достаточно хорошо представляют себе, так сказать, «матчасть».

И все бы хорошо, если бы речь шла о небольших программах. Но для банковских и биллинговых систем, включающих в себя массу разнородного «железа» и программных компонентов, такого рода инструментарий становится неадекватным в силу некоторых неприятных свойств, о которых расскажем далее.

Неполнота и неадекватность

Системы диагностики, разработанные параллельно с диагностируемым ПО, «заточены» на проблемы, выявленные в ходе разработки. Часто к моменту ввода системы в эксплуатацию многие из них бывают решены, а соответствующие тесты теряют актуальность. Работа системы в «боевом» режиме выявляет новые проблемы, но диагностический аппарат для них разрабатывать уже некому.

Система в состоянии «боевой» эксплуатации живет в условиях, сильно отличающихся от тестовых. Причем это может касаться не столько масштабов нагрузки, сколько масштабов критичности тех или иных функций. В описанной выше истории с системой сбыта в HP основные проблемы возникли, когда, в отличие от этапа лабораторного тестирования, пользователями системы оказались обычные люди, неподготовленные потребители продукции, которым не удавалось самостоятельно сформировать необходимый заказ. Менеджеры компании, планировавшие реформу системы сбыта, не сумели заранее оценить масштаб проблемы и предусмотреть роковые последствия.

Неоднородность и неинтегрируемость частей друг с другом

Различные части большой прикладной системы разрабатываются разными людьми, а зачастую и разными компаниями, при этом используются разные программные продукты, библиотеки и т.д. Естественно, совместимость между компонентами в итоговом продукте должна быть, иначе он просто не будет работать. Если те же люди создают заодно и средства диагностики, то стимулы унифицировать эти средства у них обычно отсутствуют. Результат (как в песне: «Я его слепила из того, что было») представляет собой набор разнообразных инструментов, функционирование которых, системные требования к которым, их зависимости от других программных продуктов и процедуры установки не только не укладываются в определенную систему, но чаще всего вообще не документированы. В результате такие средства невозможно объединить в единую систему даже между собой, что конечно, затрудняет (точнее, делает невозможным) отторжение этих средств от разработчика и передачу заказчику, который, вдобавок ко всему, степенью подготовки и менталитетом сильно отличается от разработчика.

Неинтегрируемость со средствами мониторинга системотехники

Заказчик прикладной системы хочет знать, насколько она отвечает требованиям его бизнеса и в какой степени справляется с его бизнес-задачами. Ему интересно, например, сколько записей об оказанных услугах обрабатывается в настоящее время его биллинг-системой, или что происходит со счетами в банковской системе, сколько записей или счетов ожидает обработки и как долго длится это ожидание. Если что-то идет не так, заказчик хочет видеть масштаб и серьезность неполадок, понимать, насколько быстро их можно устранить и как это в целом отразится на его бизнесе.

В отличие от разработчика, этому самому бизнесу в общем-то все равно, вызваны данные

неполадки ошибкой в ПО или выходом из строя кондиционера в серверном помещении, поэтому мониторинг работы программных продуктов должен осуществляться параллельно с мониторингом соответствующих аппаратных средств, а данные из обоих источников должны сходиться в какой-то точке, преобразуясь в единую картину, для бизнеса интересную и понятную.

Разработчиков же вопросы «железа» интересуют во вторую очередь, а созданные ими элементы системы диагностики не только сами не затрагивают аппаратную часть, но и не могут быть соединены с другими, предназначенными для этой аппаратной части диагностическими средствами.

Тенденциозность

Хоть и нечасто, но бывает, что при вводе системы в эксплуатацию мнения разработчиков и заказчиков о том, какое поведение этой системы считать нормальным, а какое критическим или аварийным, могут расходиться. Это может внести определенный элемент субъективности при выборе как самих диагностических параметров и их критических значений, так и алгоритмов измерений.

Ограниченные возможности представления и экспорта данных

Информация, получаемая «самодельными» средствами диагностики, предназначается исключительно для оператора. Выводится она чаще всего в окна на экране или в журнальные файлы так называемого «свободного» формата (то есть формат как таковой отсутствует). Средства представления обычно весьма ограничены и нерасширяемы.

С другой стороны, нам часто хочется не только посмотреть, что и как «чувствует» прикладная система в данный момент, но и заглянуть в прошлое, например, чтобы понять, что случилось с ней вчера вечером, накопить данные и «повертеть» их в виде графиков и диаграмм, посмотреть статистику за последний день или месяц, экспортировать все это в отчет или базу данных, агрегировать технические показатели и пересчитать их в бизнес-показатели и т.д., и т. п.

«Самодельные» средства обычно не только сами «не умеют» делать всего этого, но и не позволяют отдать данные в другую систему, ту, которая «умеет».

Преодолеть все эти недостатки призваны так называемые промышленные системы мониторинга.

Промышленная система мониторинга

Итак, мы подошли к идее приобретения промышленной системы. Некоторое количество таковых имеется в продаже, но прежде чем отправляться на рынок, надо бы сначала самим понять, чего мы ожидаем от покупки. Наша идеальная система мониторинга должна:

- иметь средства обнаружения сбоев;
- иметь средства накопления исторических данных и статистики;
- обеспечивать удобное и разнообразное представление накопленной информации;
- позволять преобразование низкоуровневых данных в «бизнес-картину» вообще и KPI в частности;
- предоставлять шлюзы в другие промышленные системы представления и обработки информации и/или API для построения такого рода шлюзов;
- иметь достаточный набор готовых модулей для мониторинга стандартного оборудования и «стандартного» ПО;
- предоставлять возможность разработки собственных модулей для мониторинга заказных приложений.

Рассмотрим подробнее, что даст выполнение этих требований.

Средства обнаружения сбоев

В случае обнаружения сбоя формируется аварийное сообщение и оповещаются все, кто отвечает за объект.

Кроме того, если способ устранения сбоя известен[6], система должна обеспечивать автоматическую или полуавтоматическую его реализацию (Recovery Actions).

Средства сбора статистики

Система должна не только показывать, что происходит «на текущий момент», но и накапливать исторические данные, что даст возможность проводить их статистическую обработку, получать усредненные характеристики качества работы прикладной системы, обнаруживать тенденции и корреляции между значениями различных показателей и сбойными ситуация-

6 К сожалению, это случается очень редко. Обычно на момент внедрения системы мониторинга нет данных о наиболее характерных неполадках и способах их устранения. Побочным эффектом внедрения «правильной» системы мониторинга может стать накопление таких знаний и придание им формы Recovery Action.

ми. Такой взгляд в прошлое позволяет в какой-то степени предвидеть будущее и предотвращать неприятности.

Средства отображения информации

Хотелось бы, чтобы система позволяла не только «по-всякому» показывать пользователям состояние прикладной системы, но делать это по-разному для разных пользователей. Например, оператору дежурной смены обычно можно и нужно видеть только часть системы, за работоспособность которой он непосредственно отвечает, причем с максимальной детализацией и возможностью запуска «корректирующих воздействий». Руководителю же дежурной смены, может быть, такая детализация не нужна, но ему необходимо видеть панораму всей прикладной системы и статистику. Бизнес-аналитикам нужны данные, «переваренные» до бизнес-терминов. Каждому должно быть доступно лишь то, что ему нужно, причем в наиболее удобном для него виде.

Средства преобразования в «бизнес-картину»

При необходимости наша система должна «уметь» пересчитывать сотни и тысячи низкоуровневых показателей в единицы и десятки ключевых параметров (KPI), необходимых для бизнеса.

Что выбрать?

Решив приобрести промышленную систему мониторинга, мы оказываемся перед проблемой выбора. Компания «Инфосистемы Джет» шесть лет занимается разработкой и внедрением систем мониторинга, используя несколько разных продуктов. Различаются они достаточно сильно, и даже поверхностное сравнение их требует отдельной статьи, к тому же часто приводит к «религиозно-маркетинговым» спорам о том, чья система лучше. Поэтому отметим лишь, что наиболее удобной для мониторинга приложений системой для нас оказалась линейка продуктов BMC Patrol.

Описание архитектуры и принципов действия продуктов этой линейки представлено далее в этой статье. Основные же их «тактико-технические» характеристики желающие могут найти по адресу: http:/www.bmc.com/products/productlist/0,,19052,00.html/. Здесь лишь подчеркнем, что определяющим фактором для нас является наличие среды разработки, полноценной в том смысле, что она позволяет делать модули, архитектурно ничем не уступающие модулям, произведенным самой компанией BMC Software.

Нужны специалисты

Итак, оценив несколько промышленных систем мониторинга, мы нашли подходящую. Однако каким бы ни оказался наш выбор, система всегда будет сложной, поэтому ее недостаточно купить, надо еще и внедрить. А это потребует специалистов, которые:

- мастерски владеют внедряемой системой;
- обладают достаточными навыками работы и администрирования различных программно-аппаратных комплексов;
- умеют собирать, оценивать и систематизировать разрозненную информацию о прикладной системе в условиях отсутствия документации и недоступности людей, обладающих необходимой информацией;
- умеют писать программы для различных программно-аппаратных платформ на разных языках программирования, смешивать при необходимости разные языки в одной программе.

Чаще всего компания, планирующая внедрение системы мониторинга, не имеет в своем распоряжении коллектива разработчиков, удовлетворяющих перечисленным требованиям, поэтому для решения этой задачи привлекает внешнюю команду, специализирующуюся на выполнении такого рода проектов.

О системотехнике, заказных приложениях и разработке ПО вообще

В зависимости от объекта мониторинг делится на две категории: мониторинг системотехники (то есть «железа», операционной системы) и мониторинг прикладных систем. Попробуем сравнить эти две разновидности поаспектно, а затем остановимся на отличиях разработки модулей

мониторинга приложений от разработки «обычных» программных продуктов. Зачем это нужно? Дело в том, что незнание или неучет особенностей мониторинга *приложений* может привести к неверной оценке возможностей и сроков реализации проектов, обманутым ожиданиям, неудовлетворению, а то и разочарованию в самой идее. Многое из изложенного ниже, возможно, покажется очевидным и недостойным упоминания, однако, к сожалению, именно тривиальности и очевидности с удивительным постоянством забываются и игнорируются заказчиками практически в каждом проекте.

Заранее оговоримся, что данное сравнение отражает исключительно опыт автора и его ближайших коллег. Этот опыт, в силу его несомненной ограниченности, безусловно, отражает лишь часть общей картины и в чем-то может не совпадать или даже противоречить опыту других специалистов.

Мониторинг системотехники и приложений будем сравнивать по направлениям:

- особенности постановки задачи;
- источники данных;
- вид работ и квалификация исполнителей;
- объект мониторинга;
- возможности отладки и тестирования.

Сравнение мониторинга приложений с разработкой обычного ПО проведем по следующим «осям»:

- особенности цикла разработки;
- особенности эксплуатации и сопровождения.

Постановка задачи

Постановка задачи системы мониторинга должна содержать ответ на вопросы: что измеряем, зачем и что делать, если возникла проблема?

Когда имеешь дело с системотехникой, ответить на первый вопрос обычно бывает легко. Параметры работы системы очевидны и/или исторически определены/стандартизованы. Например, работа компьютера характеризуется такими параметрами как загрузка процессора, заполнение памяти и файловых систем и т.д.

Другая особенность заключается в том, что для системотехники определение первопричины явлений чаще всего находится за пределами возможностей системы мониторинга. Например, если обнаружена повышенная загрузка процессора, максимум, что может сделать система, это определить процесс, ответственный за эту загрузку. Система мониторинга ничего не знает о задаче, выполняемой этим процессом, следовательно, не знает, является ли такая загрузка действительно «ненормальной», и тем более не знает, что конкретно привело к ее повышению и как с этим бороться.

Поэтому ответ на вопрос «зачем измеряем?» носит общий характер, а ответ на «что делать в случае проблемы?» отсылает вопрошающего к «системному администратору».

В мониторинге приложений, напротив, номенклатура измеряемых величин первоначально неизвестна, и ее выработка представляет собой сложную задачу. В отличие от мониторинга системотехники, мы должны не только получить достаточно четкое представление о том, что именно измеряем, но обязательно знать, как измеренное значение характеризует работу приложения (зачем измеряем?), что является признаком отклонений, и какие действия должен произвести оператор в случае их обнаружения (что делать?)[7].

Жизнь такова, что на момент планирования системы мониторинга значительная часть этой информации отсутствует, поэтому сбор ее обычно продолжается на этапах разработки, тестирования и эксплуатации.

Источники исходных данных

Работоспособность системотехники определяет жизнеспособность бизнеса, поэтому при достаточно жесткой конкуренции на рынке и относительно широком выборе товара никому не придет в голову ни продавать, ни покупать дорогое оборудование или так называемое mission-critical ПО без достаточного документального обеспечения, действенной службы поддержки, системы обучения и сертификации специалистов и т.д.

Таким образом, информационно мониторинг системотехники обычно поддержан достаточно хорошо, и если необходимость построения какого-то модуля все-таки возникает (для стандартных решений обычно существуют уже готовые модули мониторинга), то он может быть создан на основе существующих источников информации.

Заказные же программные системы — товар штучный, уровень документирования и под-

7 Зачастую заказчик хочет мерить что-то, названия для чего просто нет (например, показывает несомненно полезный SQL-запрос, но затрудняется объяснить, как интерпретировать возвращаемое значение). В результате по ходу постановки задачи мониторинга приходится иногда даже создавать новую терминологию, относящуюся к самой прикладной системе.

держки, доступный при массовом производстве, чаще всего недостижим для штучного. Поэтому оказывается, что основной источник информации здесь — люди, а именно:

- разработчики приложения;
- команда внедрения;
- специалисты заказчика.

Разработчики прикладной системы

Разработчики прикладной системы являются одновременно наиболее ценным и наименее доступным ресурсом. Они знают все о разработке, причем, что немаловажно, информация эта достоверна и актуальна. Но, к великому сожалению, система мониторинга обычно не входит в круг их интересов и прямых обязанностей, к тому же они нередко удалены географически, поэтому получение аудиенции у кого-то из них можно рассматривать как победу[8].

Итак, разработчики:

- знают много;
- их информация достоверна;
- в мониторинге не заинтересованы;
- практически недоступны.

Команда внедрения

Случается, что внедрение заказных систем выполняется их разработчиками, но все же чаще для этого нанимается отдельная команда. В силу непосредственной близости к объекту мониторинга люди эти более доступны для контактов, чем разработчики, они могут оказаться полезным источником информации, но по сравнению с разработчиками, их информация более скудная и менее актуальная.

Команда внедрения, хотя и работает с объектом ежедневно, получает знания «из вторых рук», причем зачастую разработчики для них тоже малодоступны.

Обычно именно через этих людей мы получаем доступ к письменным «первоисточникам», написанным когда-то разработчиками, но часто утратившим актуальность. В условиях недостатка «официальной» документации команда внедрения пишет свои собственные документы и инструкции, опираясь во многом на собственные наблюдения и гипотезы разной степени достоверности.

Иногда люди эти в определенной степени заинтересованы в системе мониторинга, понимая, что она могла бы оказаться полезной для их работы, они охотнее разработчиков идут на контакты, участвуют в обсуждении постановки задачи и способов ее реализации. Но из-за того, что система мониторинга не входит в их прямые обязанности, контакты эти приходится осуществлять «в свободное от работы время», которого у них обычно нет.

Итак, команда внедрения:

- информация получается «из вторых рук»;
- информация частично устаревшая, степень актуальности, как правило, неизвестна;
- в проекте заинтересованы слабо;
- относительно доступны.

Специалисты заказчика

Специалисты заказчика, по определению, являются стороной, наиболее заинтересованной в мониторинге. Именно для них предназначается разработка, именно им потом «с ней жить», поэтому именно они наиболее охотно делятся информацией. К сожалению, именно они оказываются и наиболее скудным ее источником. По отношению к прикладной системе специалисты заказчика выступают в роли пользователей, а за все остальное спрашивают с команды внедрения. Собственной документации у них нет, и все что есть, получено от группы внедрения или от разработчиков.

Можно, конечно, помечтать о том, что в идеале для создания системы мониторинга у заказчика должна быть создана рабочая группа, которая возьмет на себя сбор исходных данных и постановку задачи. Разработчики системы мониторинга должны получать информацию непосредственно из их рук, а не искать прямых контактов с разработчиками прикладной системы или командой внедрения, но, к сожалению, в нашей практике с такой ситуацией встречаться пока не приходилось.

Итак, специалисты заказчика:

- в проекте мониторинга *заинтересованы*;
- доступны для контактов;
- информации мало;
- она часто устаревшая.

Общий вывод: получение информации, необходимой для реализации системы мониторинга заказной прикладной системы, является наиболее *трудоемкой и продолжительной по времени* задачей, во многом определяющей трудоемкость и продолжительность осуществления всего проекта.

8 Мои собственные победы такого рода можно пересчитать по пальцам одной руки, а однажды за полтора часа общения с разработчиком, в ходе которого необходимая мне информация извлекалась непосредственно из исходных текстов, удалось получить данные, обеспечившие меня работой на целый месяц!

Виды работ и квалификация исполнителей

Типовая для мониторинга системотехники задача — наблюдение за множеством однотипных рабочих мест (например, компьютеров с WindowsXP). Работы включают в себя освоение и массовую установку готовых программных продуктов, общение с пользователями, с одной стороны, и службами поддержки производителей продуктов, с другой. Эти задачи рассматриваются как составные части процесса администрирования и более или менее вписываются в рамки навыков и обязанностей штатных системных администраторов.

Мониторинг приложений представляет собой прежде всего создание нового *программного продукта*, что требует усилий *программистов*, а не системных администраторов.

Объекты наблюдения

Системотехника как объект мониторинга изменяется относительно редко, причем версии продуктов качественно разделены, изменения документированы, документация эта доступна.

Заказные же приложения имеют тенденцию постоянно меняться, исправление ошибок разработчиками приводит к потоку патчей или (что бывает чаще) новых версий, которые, в свою очередь, содержат новые ошибки. Информация же о том, что именно изменилось, практически отсутствует.

Система мониторинга является по отношению к прикладной системе внешней, поэтому в идеальном мире взаимодействие между ними всегда основано на некотором наборе соглашений, называемом протоколом или API. Протокол этот специально разрабатывается, фиксируется и документируется, изменения в нем производятся достаточно редко, все заинтересованные стороны о них своевременно уведомляются. Ответственность за все это несет разработчик прикладной системы. Лишь если все названные условия выполнены, система мониторинга годами работает бесперебойно без вмешательства ее разработчиков, в то время как прикладная система находится в развитии.

В реальности добиться от разработчиков фиксации отдельного API для мониторинга удается далеко не всегда. О «фатальных» модификациях прикладной системы мы узнаем по отказу тех или иных модулей мониторинга, которые приходится постоянно дорабатывать. Без авторского надзора система мониторинга в этом случае довольно быстро «стареет» и «умирает».

Отладка и тестирование

Когда задумывается система мониторинга, этот аспект разработки вообще обычно выпадает из рассмотрения. Речь идет о том, где, как и на чем отлаживается и проверяется работа создаваемых программных модулей.

Отладка мониторинга объекта системотехники проводится, естественно, на этом самом объекте, который производится обычно в массовых количествах, поэтому доступен для покупки или аренды. Проще говоря, для разработки и отладки мониторинга имеющегося у заказчика объекта X, достаточно раздобыть (купить/арендовать/взять у производителя под разработку) точно такой же объект X и производить отладку на нем. Вопрос технический и обычно решаемый.

В случае прикладных систем дело обстоит гораздо хуже. Они уникальны. Не в смысле исключительного качества исполнения, а в том смысле, что существует такая система в единственном экземпляре, причем находится у заказчика и работает в «боевом» режиме. Установить второй экземпляр где-то еще для отладки и экспериментов невозможно по ряду объективных и субъективных причин. Организовать отдельное рабочее место вблизи самого объекта мониторинга не всегда возможно, да и сам объект может находиться за тысячи километров. Влезать лишний раз в «боевую» систему со своим не отлаженным еще диагностическим продуктом боязно, неприятные последствия неверного движения могут быть масштабными.

Кроме того, некоторые из диагностируемых процессов могут происходить в прикладной системе достаточно редко, раз в сутки или раз в месяц, что тоже существенно затрудняет тестирование.

Можно, конечно, как «боксер-заочник», совсем отказаться от отладки, надеясь на свою абсолютную безошибочность, но путь этот — для сильных духом, поэтому чаще все же разработчику приходится своими руками строить отладочный стенд, микро-приложение, имитирующее процессы в «настоящей» системе в пределах, необходимых для отладки модулей мониторинга.

Необходимость построения такого стенда часто вообще не учитывается при планировании работ, но время, затраченное на него, как правило, сравнимо с продолжительностью разработок самих модулей.

Цикл разработки

Система мониторинга приложения сама является программным продуктом. Теоретически разработка любого программного продукта организуется в виде проекта, то есть имеет определенное начало, четко различимые этапы и определенные условия завершения. Последовательно исполняемые этапы проекта должны включать в себя сбор информации, постановку задачи, планирование последовательности и сроков выполнения работ, непосредственную реализацию, внедрение, разработку документации, сдачу/приемку и сопровождение.

Практика построения систем мониторинга диктует несколько иную картину.

Во-первых, если отдельные модули практически любой прикладной программной системы по необходимости тесно завязаны друг на друга (результат работы одного модуля обычно является источником данных для другого), то модули мониторинга логически привязаны лишь к «своим» объектам наблюдения, а между собой связаны достаточно слабо. Если еще учесть, что система мониторинга должна быть более «живучей», чем наблюдаемая прикладная система, то модули мониторинга не только могут, но и должны функционировать автономно, чтобы отказ одного модуля не привел к отказу другого.

В силу этой независимости последовательность разработки и внедрения модулей в рамках одного проекта может быть произвольной и определяться внешними условиями, например, пожеланиями заказчика или тем, как идет сбор исходной информации.

В результате на практике разработка системы мониторинга (вне зависимости от того, как она была запланирована) из единого проекта превращается в несколько отдельных мини-проектов.

Сопровождение

Работа над системой мониторинга не заканчивается сразу же после внедрения. Причем сопровождение системы мониторинга не сводится лишь к традиционным консультациям и исправлению ошибок. Именно на этом этапе появляются ответы на многие вопросы, которые не были найдены на этапах планирования и разработки.

Определение/уточнение пороговых значений

Если ассортимент измеряемых параметров был определен при планировании системы, то их пороговые значения на этом этапе часто оценке не поддаются, поскольку параметры эти никто до сих пор не измерял, и значения их известны не были. Чаще всего эти «пороговые» величины могут быть определены лишь в процессе эксплуатации[9].

Коррекция методов измерения

Как уже было упомянуто выше, прикладные программные продукты являются весьма подвижным объектом наблюдения. Смысл «правильно» определенных параметров измерения непосредственно связан с соответствующими характеристиками работы прикладной системы и, как правило, не меняется. Зато методы их измерения чаще всего определяются *реализацией* прикладных модулей (кроме редких случаев, когда разработчик предоставляет API для мониторинга). Изменения в прикладной системе обычно проявляются в системе мониторинга в виде сообщений о невозможности проведения измерений или подозрительных значений параметров. Следовательно, чтобы система мониторинга работала, пользователи должны своевременно извещать ее разработчиков обо всех подозрительных явлениях, а последние, в свою очередь, оперативно производить соответствующие ремонтные работы.

Интересно, что как аппетит появляется во время еды, так и реальный интерес к системе мониторинга приходит, когда системой начинают пользоваться. Когда человек получает возможность проводить над прикладной системой реальные тесты и измерения, он, так сказать, входит во вкус и действительно задумывается, какую пользу можно извлечь из системы. В целом это относится к любым программным продуктам, но в системах мониторинга проявляется наиболее ярко.

Поэтому на этапе эксплуатации активность и любознательность пользователя необходимо всячески стимулировать. Надо максимально облегчить ему формулирование и формализацию наблюдений и пожеланий, а также привязку их к параметрам, модулям и другим объектам системы мониторинга. Хороших результатов можно достигнуть, если дать пользователю справочную систему, которую он сам может корректировать и дописывать. На момент сдачи в эксплуатацию такой справочник может содержать комплект документации, который помимо своего основного назначения служит еще и опреде-

9 Поступление такого рода информации от сотрудников заказчика после сдачи системы мониторинга в эксплуатацию — верный признак того, что она реально используется.

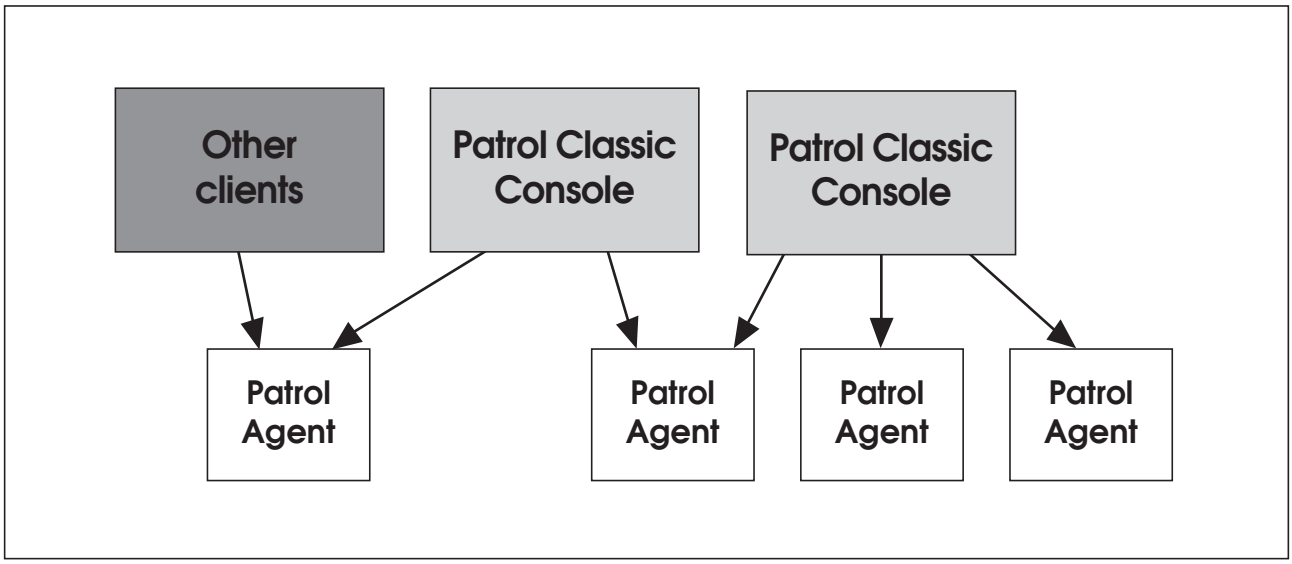

Рис. 1 Архитектура Patrol Classic

ленным образцом, как добавлять информацию. В процессе эксплуатации справочник дополняется определениями пороговых значений, описанием ситуаций, при которых происходили различного рода сбои, перечислениями возможных действий по их устранению, а также комментариями относительно работы как прикладной системы, так и системы мониторинга[10].

Со временем, когда такой привнесенной пользователями информации накопится много, она может стать основой дальнейшего развития системы мониторинга.

BMC Patrol и мониторинг приложений

В первой части статьи уже упоминалась эта линейка продуктов BMC, теперь остановимся на ней подробнее. Сразу оговоримся, что материал изложен с точки зрения разработчика модулей мониторинга, поэтому рассматривается лишь часть продуктов, которая относится к категории *Patrol Classic*.

BMC Patrol представляет собой распределенную многоуровневую систему. Этот продукт позволяет строить как простые решения (клиент-сервер), так и трехуровневые системы с централизованным управлением, охватывающие тысячи объектов.

Существуют два варианта построения системы на основе BMC Patrol. В терминологии самой компании BMC первый называется «классическим» (Patrol Classic), он же — Patrol третьей версии или просто Patrol3. Исторически эта модель была первой, на ее основе можно строить небольшие системы, управляющие небольшим количеством компьютеров с небольшого количества операторских мест.

Второй вариант, появившийся позднее, называется Patrol Central или Patrol седьмой версии (Patrol7)[11]. Для сбора и накопления информации в этом варианте используются те же классические агенты, но добавляется дополнительный инфраструктурный слой, позволяющий построение систем с централизованным управлением, и количество агентов уже может измеряться тысячами.

Разработчик модулей мониторинга имеет дело, в основном, с компонентами классической модели, поэтому здесь Patrol7 рассмотрен не будет.

Классический Patrol построен по ставшей уже традиционной схеме «клиент-сервер», компонентами которой (см. рис. 1) являются агенты (PATROL Agent) и классические консоли (Patrol Classic Console). Агенты в этой схеме являются серверами, тем местом, где собирается и накапливается информация. Консоли же это клиенты,

10 Мы разработали такую справочную систему на основе MediaWiki http://www.mediawiki.org/wiki/MediaWiki

11 В литературе можно найти упоминание четвертой версии продукта, являющейся, по-видимому, промежуточной и предтечей Patrol Central. Неизвестно, что произошло с пятой и шестой версиями.

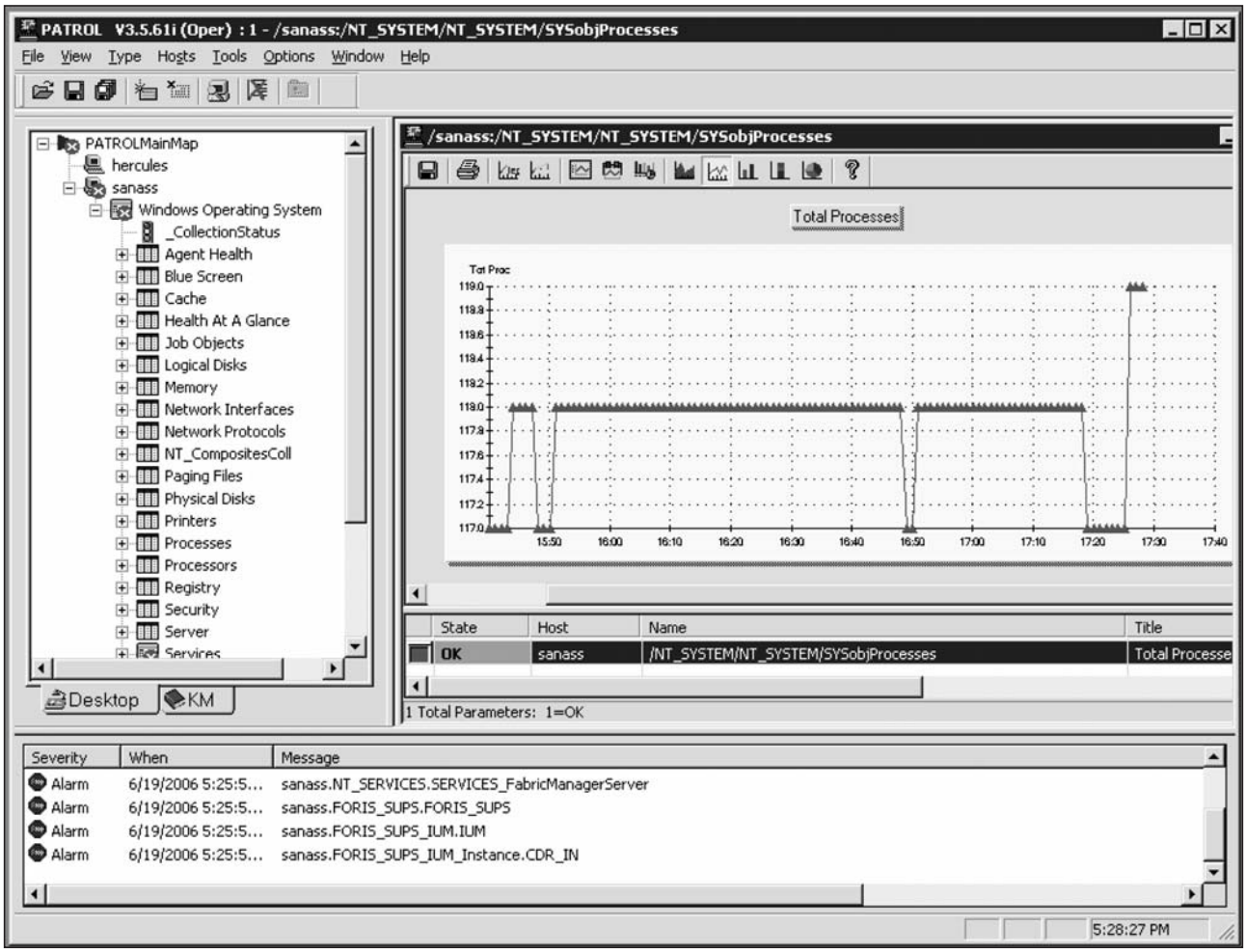

Рис. 2. Patrol Classic Console for Windows

рабочие места операторов и разработчиков. Они снабжены оконным интерфейсом и используются одновременно и для представления информации, собранной агентами, и для управления самими агентами.

Отметим, что кроме классической консоли к агентам могут подключаться другие приложения линейки Patrol, не описанные в этой статье. Имеется также API, позволяющий разрабатывать такие программы «третьими лицами», что открывает широкий простор интеграции.

Patrol Agent

Агент является ключевым элементом системы BMC Patrol. Он собирает и хранит информацию. Агенты самодостаточны и автономны, будучи раз настроенными, они могут работать неограниченное время[12].

Агент осуществляет следующие функции:

- выявление объектов для наблюдения;
- сбор численных метрик на регулярной основе;
- хранение истории изменения метрик;
- интеллектуальное взаимодействие с консолями и рассылка данных по подписке;
- рассылка оповещений о событиях (events);
- запуск корректирующих действий (recovery actions).

Для сбора численных метрик в Patrol используются так называемые параметры (parameters). Параметры ассоциируются с динамически создаваемыми объектами, что позволяет наблюдать за произвольным числом объектов, например, за всеми процессами операционной системы или всеми файловыми системами, динамически отслеживая их количество.

Такая интеллектуально-автономная архитектура вкупе с весьма экономной рассылкой данных по подписке (описана ниже) позволяет производить всю специфическую работу в непосредственной близости от объекта измере-

12 Этим Patrol отличается, например, от системы HP OpenView, в которой агенты являются лишь «датчиками» для работы которых требуется соединение с центральным сервером управления. куда они передают собранные данные.

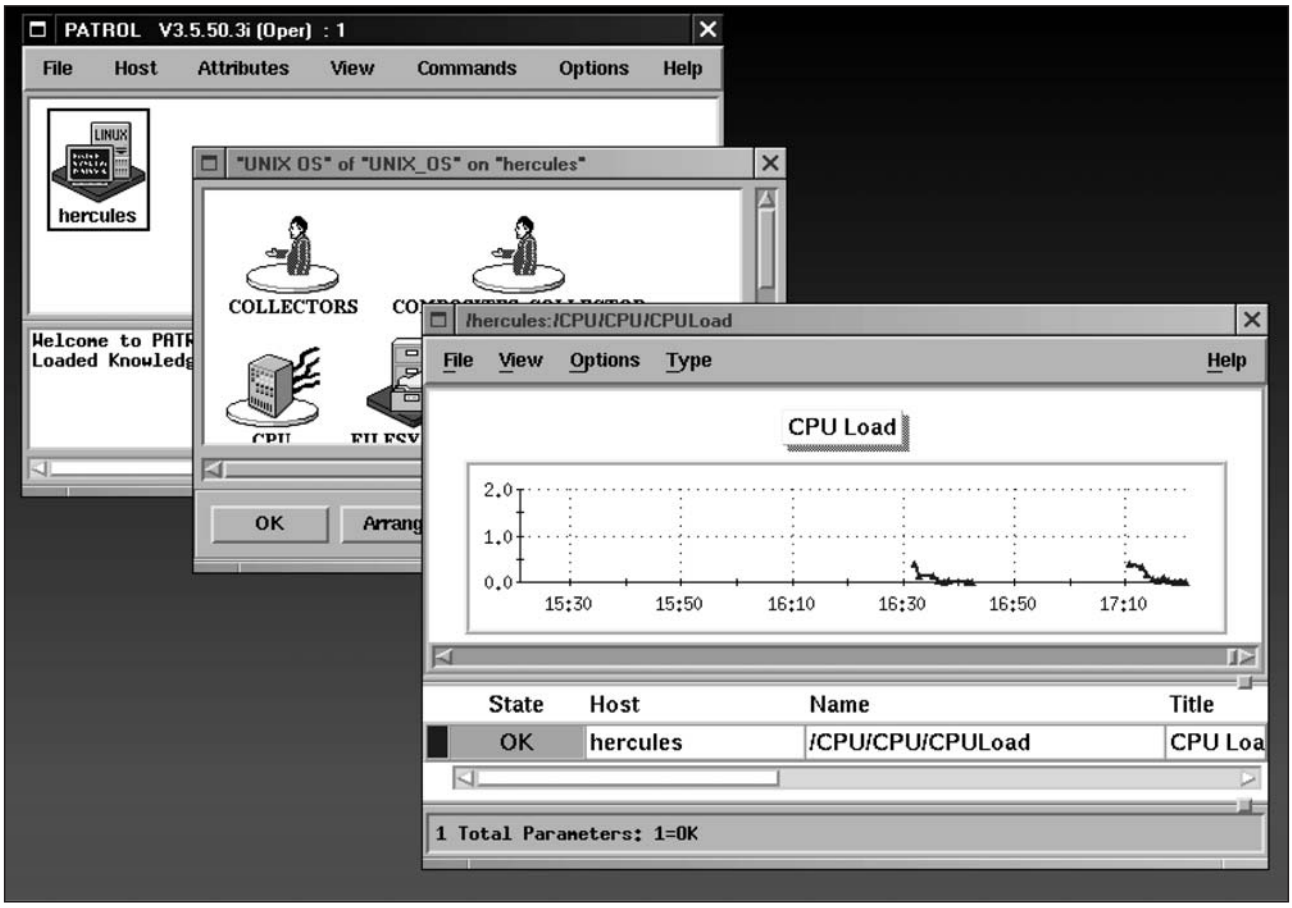

Рис. 3. Patrol Classic Console for Unix

ния, что снижает нагрузку на сеть и смежные системы обработки.

Это позволяет масштабировать систему мониторинга на тысячи компьютеров, так как необходимые для сбора и первичной обработки данных вычислительные мощности при этом автоматически равномерно распределяются по всей системе[13].

Patrol Classic Console

Для доступа к собранной агентами информации используются клиентские приложения, так называемые *консоли* (Patrol Consoles), соединенные с агентами по сетевым протоколам, основанным на TCP и UDP. В архитектуре Patrol3 консоли соединяются с агентами непосредственно, причем приложения этого типа так и называются Patrol Classic Console.

К функциям консоли относятся:

- подписка на информацию об объектах *desktop tree* (см. ниже), отображение этого дерева;
- получение истории изменения значений параметров и их отображение на экране в виде графиков;
- получение и отображение событий (events), рассылаемых агентами;
- поддержка интерактивного взаимодействия пользователей с приложениями мониторинга;
- разработка модулей мониторинга (в режиме разработчика).

Консоли «для Unix» и «для Windows»

Существуют две разновидности классической консоли — в реализации для Microsoft Windows (рис. 2) и для Unix (рис. 3).

Хотя по исполняемым функциям эти приложения достаточно близки, на иллюстрациях хорошо видно, что их пользовательские интерфейсы отличаются кардинально[14].

13 Однако управление тысячами агентов, а также использование собранных ими данных может представлять в «классической» модели определенные сложности, для преодоления которых создавалась система Patrol7.

14 Соответственно, существует два отдельных комплекта документации, что несколько затрудняет обучение пользованию продуктом.

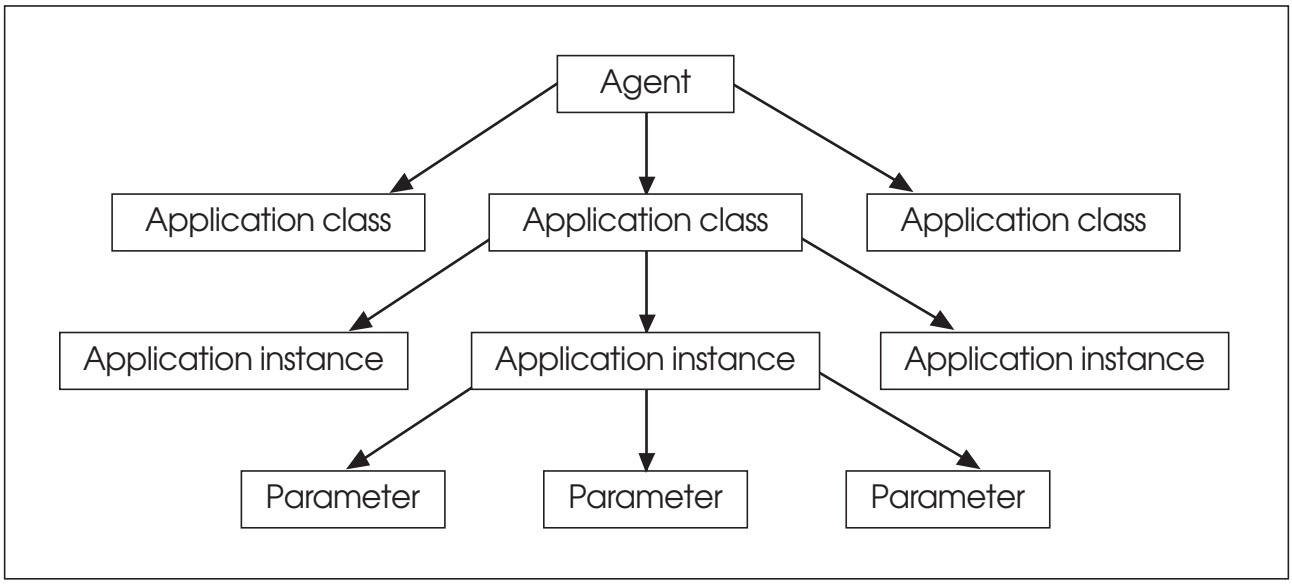

Рис. 4. Patrol Agent Object Tree

Режимы «оператор» и «разработчик»

Классическая консоль может работать в двух режимах — оператора (Operator) и разработчика (Developer). В режиме оператора пользователю доступны лишь функции наблюдения, а режим разработчика позволяет «на ходу» модифицировать модули мониторинга в агенте, а также выполнять ряд дополнительных управляющих воздействий, вроде остановки или перезапуска агента, полезных исключительно для разработки или отладки[15].

Категории данных

В данном разделе описаны структуры данных, которые агент использует для организации, и консоль — для представления результатов измерений.

Дерево объектов агента

Значительная часть динамических данных, которыми манипулирует агент, представляет собой иерархическую древовидную структуру, здесь и далее называемую *деревом объектов агента*. Каждый элемент этого дерева может принадлежать одному из следующих классов:

- параметры (parameter instance);
- экземпляры приложений (Application instance);
- классы приложений (Application class);
- переменные (Object variable).

С *параметрами* связываются численные результаты измерений, а также история изменения их значений за определенный период времени. Каждый параметр принадлежит определенному *экземпляру приложений* (Application instance), а тот, в свою очередь, относится к определенному классу приложений (Application class). Классом приложения определяются свойства его экземпляров, включая состав и свойства параметров.

В дереве объектов агента классы приложений находятся наверху, под классами — экземпляры, под ними — параметры (см.рис. 4). «Корнем» дерева является сам агент.

Необходимо подчеркнуть, что изображенная на рисунке схема не является лишь умозрительным представлением данных. Каждый элемент диаграммы обладает набором определенных характеристик и имеет физическое представление в памяти агента.

Для всех элементов дерева определены понятия *имени элемента* и *пути*, которые образуют *пространство имен агента* (agent namespace). Пути аналогичны полным именам файлов в файловой системе Unix, каждый путь состоит из имени элемента и имен вышестоящих элементов, вплоть до корневого, имя которого стоит первым, разделенных символом «/».

Путь к каждому параметру включает в себя три фрагмента: имя класса приложений, имя экземпляра приложения и имя параметра. Путь

15 «Представительские» возможности консоли в Patrol7 (Patrol Central Console) значительно перекрывают возможности классической консоли, поэтому, если есть выбор, оператору лучше пользоваться более продвинутой центральной консолью. Однако классическая консоль является единственным штатным средством создания приложений мониторинга, поэтому разработчику приложений без нее никак не обойтись.

«/CPU» – класс приложений CPU

«/CPU/CPU1» – экземпляр с именем CPU1 класса CPU

«/CPU/CPU1/CPUUtil» – параметр CPUUtil экземпляра CPU1 класса CPU

Рис. 5. Примеры путей к объектам

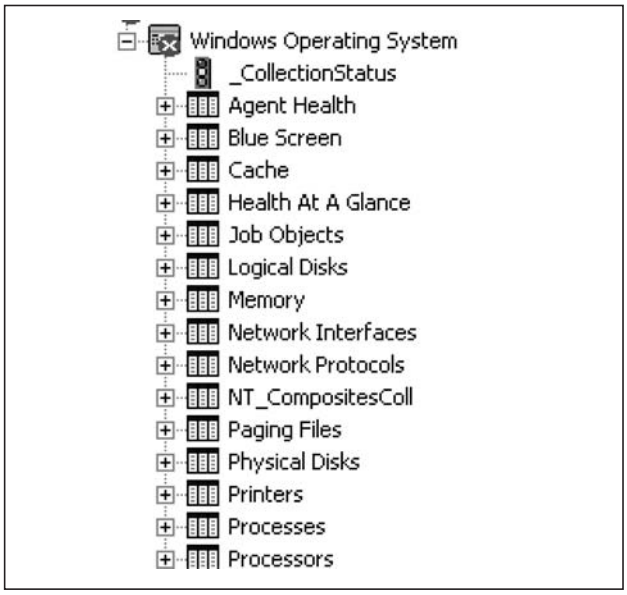

Рис. 6. Patrol Console Desktop Tree

к экземпляру приложения, соответственно, только имя класса и экземпляра, а путь к классу – только имя класса. Все пути начинаются с символа «/», которое неявно принимается за имя агента (см. рис. 5).

Для каждого элемента дерева определен набор *переменных*, которые принимают текстовые значения. Как и объекты, переменные именованы, для доступа к ним также используются пути. Часть переменных являются *встроенными*, их значения могут вычисляться динамически в момент запроса.

С деревом объектов приходится обычно сталкиваться разработчику, который имеет непосредственное отношение к построению этого дерева, а пути использует для адресации узлов. Оператор же обычно видит в консоли другое дерево – *desktop tree*.

Дерево объектов консоли

Дерево объектов консоли (в терминологии BMC оно называется *desktop tree*) – это «презентационное» представление данных, предназначенное для конечного пользователя системы, т.е. для оператора. Состоит оно только из экземпляров приложений и параметров. Параметры, как и в дереве объектов, принадлежат к экземплярам, но вот экземпляры «растут» уже не из классов приложений, а непосредственно из узла агента или из других экземпляров.

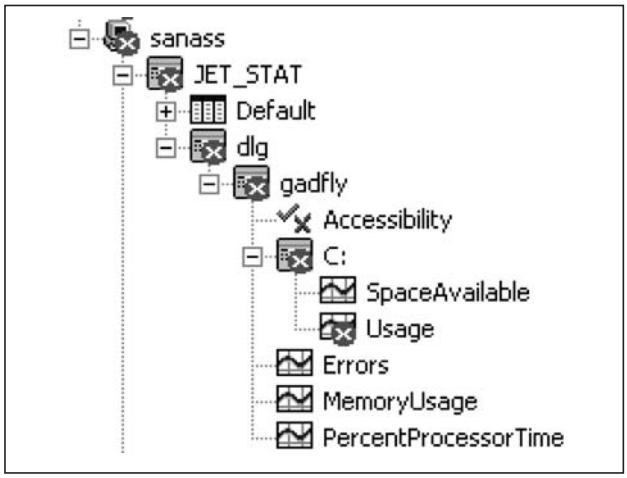

Рис. 7. Patrol Console State Propagation

Пример фрагмента такого дерева представлен на рис. 6. Здесь узел CollectionStatus соответствует параметру, а остальные узлы – экземплярам приложений. Экземпляр Windows Operating System является «структурообразующим», у него нет параметров, его цель – сгруппировать другие ветви дерева.

В отличие от дерева объектов агента структура *desktop tree* лишь частично определяется типами составляющих элементов. В момент создания каждого экземпляра явно указывается, какой экземпляр является для него «материнским». Эта структура произвольна и выбирается разработчиком, исходя из удобства представления данных. Например, если агент установлен на компьютере и измеряет три характеристики одного центрального процессора, то для представления этой информации достаточно одного элемента приложения с тремя параметрами. Если же процессоров в системе несколько, и мы хотим иметь информацию по каждому из них отдельно, то нам потребуется по элементу приложения на каждый, причем количество этих элементов заранее неизвестно и даже может ме-

няться динамически. Чтобы отличить эти элементы от элементов других типов, мы обычно помещаем их в отдельную ветку дерева, создав для этого один дополнительный элемент[16].

С каждым узлом desktop tree связывается так называемое состояние (status). Основными являются три состояния — OK (все нормально), WARNING (не все нормально) и ALARM (все плохо). Состояние может быть внутреннее (ruleState) и наведенное (worstParamState). Внутреннее состояние является настоящим свойством объекта. Оно может измениться автоматически при наступлении определенных условий, например, параметр может перейти в состояние WARNING или ALARM в момент, когда его значение превысит пороговую величину. Внутреннее состояние узла может быть явно установлено программно.

Наведенное состояние может зависеть от положения узла в desktop tree и выбирается как наихудшее из собственного состояния и наведенных состояний всех дочерних элементов. Именно наведенное состояние отражается в консолях в виде мигающих иконок угрожающего вида. Таким образом, переход любого узла в состояние WARNING или ALARM вызывает цепное переключение наведенных состояний вплоть до вершины дерева (в терминологии Patrol это свойство называется State Propagation). Предполагается, что таким образом можно привлечь внимание оператора, даже если сам «плохой» объект перехода в данный момент на экране не виден.

На рис. 7 видно, что параметр Usage находится в состоянии ALARM (красный кружок с крестиком на иконке), что вызвало аналогичную подсветку всех вышестоящих элементов дерева.

Чтобы консоль отражала реальное состояние *desktop tree*, необходима определенная синхронизация между агентом и консолью. Разработчики системы предусмотрели для этого очень интересный алгоритм.

Каждая консоль «подписывается» у агента на получение данных по экземплярам определенных классов, из которых состоит *desktop tree*. Пользователь может по своему усмотрению раскрывать и закрывать ветви в изображении дерева на экране консоли. В момент раскрытия консоль посылает агенту запрос относительно содержимого данной ветви, в ответ агент присылает минимальную информацию, необходимую для отображения этой ветви в раскрытом виде. Агент всегда знает о том, как в данный момент выглядит *desktop tree* в консоли, т.е. какие ветви раскрыты и, следовательно, какие элементы могут быть видны пользователю. В дальнейшем агент оповещает консоль лишь об изменениях, влияющих на отображение именно этих видимых элементов, например, о создании нового элемента в раскрытой ветви или изменении статуса существующего узла. Таким образом, сетевой обмен между агентом и консолью сводится к *абсолютному минимуму*[17].

История параметров

Основной вид измеряемых агентом данных — значения параметров. Хотя параметры могут быть произвольным текстом, чаще все же измеряются числа. Только для чисел можно установить пороги и сохранять историю.

История параметра представляет собой данные обо всех его изменениях за определенный промежуток времени. История хранится локально, в собственной базе данных. Для каждого типа параметров может быть задан свой срок хранения, а устаревшие данные из истории автоматически удаляются.

В консолях история параметров доступна в виде графиков. При необходимости данные также могут быть экспортированы в другие приложения.

События

Другим видом данных, рассылаемых агентами, являются извещения о событиях (events). События могут инициироваться как автоматически (например, при превышении параметрами пороговых значений), так и программно. Обычно все консоли оповещаются обо всех событиях, но пользователь консоли может устанавливать в агенте фильтр, явно ограничивающий номенклатуру таких событий, что может, соответственно, сократить сетевой траффик.

История событий хранится агентом в локальной базе и может быть извлечена оттуда как для отображения в консолях, так и для экспорта в другие программные продукты.

16 Наличие деревьев двух видов часто приводит к некоторой путанице. Дерево desktop tree существует исключительно в консолях, т.е. на клиентской стороне, и не имеет никакого отношения к пространству имен агента. Пути, составляющие это пространство, не зависят от структуры элементов в desktop tree.

17 Следствием того, что агент вынужден не только поддерживать свои структуры данных, но следить за их отображением на всех консолях, является дополнительная нагрузка на компьютер агента, ненужная непосредственно для измерений. Вероятно, это было одной из причин создания трехзвенной модели Patrol7, в которой агент обычно поставляет данные лишь одному клиенту (Console Server), который уже обслуживает все обычные консоли.

Конфигурационная база

База конфигурации (patrol config) — третья по счету локальная база данных агента. Хранящиеся в ней атрибуты определяют параметры работы самого агента и работающих под его управлением модулей мониторинга. База эта служебная, представляет интерес для разработчиков и администраторов и никак не проявляет себя в консолях.

Мониторинг глазами разработчика

Рассмотрим теперь, как реализуются измерения.

Агент как виртуальная машина

Агент Patrol имеет собственную виртуальную машину, под управлением которой одновременно выполняется множество процессов, то есть является системой в системе. Процессы — это результат компиляции программ, написанных на специально разработанном для этой цели языке под названием Patrol Scripting Language (PSL).

Patrol Scripting Language (PSL)

По виду PSL более всего напоминает язык C, хотя на этом, пожалуй, сходство заканчивается. Язык является интерпретируемым, единственным типом данных является текст[18].

PSL включает в себя около 200 встроенных функций. Часть из них предназначена для организации интерфейса с внешним миром (что существенно для измерений), часть — для работы с внутренними данными агента и консолей.

Обращение во внешний мир из PSL ограничено следующими способами:

- чтение и запись файлов;
- сетевые операции через TCP;
- SNMP;
- запуск процессов операционной системы и взаимодействие с ними.

Наиболее универсален последний способ. Запустив процесс операционной системы, PSL программа получает доступ к каналам его стандартного ввода и вывода, через которые может передавать процессу команды и получать результаты их исполнения. Этим способом реализуется большинство измерений, от чтения содержимого каталога файловой системы до осуществления сложных запросов к реляционным базам.

Наиболее важными функциями работы с внутренними структурами данных агента являются создание и удаление экземпляров приложений, поскольку лишь с их помощью можно формировать оба описанных выше дерева объектов. Кроме того, важна функция, позволяющая устанавливать значения параметров, с ее использованием формируется история параметров.

Knowledge modules

Все измерения в Patrol производятся при помощи так называемых модулей знаний (Knowledge modules или KM).

При ближайшем рассмотрении модули знаний оказываются конфигурационными файлами, включающими в себя фрагменты программного кода. Каждый такой модуль содержит описание одного класса приложений (см. раздел «Дерево объектов агента»).

Несмотря на то, что файлы эти имеют текстовый формат, ручное их редактирование не предусмотрено, а единственным штатным средством их создания и модификации является описанная выше консоль в режиме разработчика.

Наиболее важные структурные элементы KM — это описания параметров, KM полностью определяет их возможный состав. Наиболее важные программные единицы — скрипты, называющиеся pre-discovery и discovery, а также команды параметров.

Скрипты pre-discovery и discovery пишутся на PSL, они осуществляют одну из важных функций Patrol — определение наличия и анализ состава объектов наблюдения. Обычно модули KM находятся в выделенном каталоге и загружаются агентом при старте. При этом может оказаться, что объект измерений для какого-то модуля на данном компьютере отсутствует, или модуль вообще предназначен для другой операционной системы. Вместо того чтобы пытаться придумать какую-то заданную систему атрибутов в KM, позволяющих агенту самому отбрасывать «неактуальные» модули, разработчики Patrol пошли по иному пути. Перед тем как загрузить весь модуль, загружается и запускается только его программа pre-discovery, цель которой — определить, нужен ли вообще этот модуль в данной системе. В случае отрицательного ответа модуль выгружается, и повторных попыток загрузить его больше не предпринимается. На содержимое скрипта pre-discovery не накладываются никакие ограничения, поэтому способ этот весьма универсален.

У скрипта discovery иная функция — определить состав объектов мониторинга и скор-

18 В этом PSL схож с популярным скриптовым языком програмирования TCL, однако, в отличие от него, PSL является «закрытым», он не может быть расширен путем подключения модулей, написанных на других языках

ректировать соответствующим образом дерево объектов. Например, если модуль предназначен для наблюдения за смонтированными файловыми системами, а для отображения состояния каждой файловой системы используется свой экземпляр приложения, то задача скрипта discovery — определить текущий состав файловых систем и сличить его с текущим набором соответствующих им экземпляров в дереве агента, после чего добавить при необходимости новые объекты и удалить устаревшие. Поскольку состав объектов мониторинга непостоянен, в отличие от pre-discovery, скрипт discovery запускается агентом периодически.

Теперь настало время сказать о том, кто же производит собственно измерения и устанавливает значения параметров. Этой цели служат *команды параметров*[19]. Так же как и скрипты discovery, команды параметров запускаются периодически, причем период может определяться индивидуально для каждого класса параметров.

Не все параметры могут иметь команды. Еще интереснее, что некоторые параметры не могут иметь значений. В зависимости от возможности иметь команды и значения, параметры делятся на три типа. К первому типу относятся так называемые стандартные (STANDARD) параметры. У них могут быть и значение, и команда. Ко второму типу относятся параметры-потребители (CONSUMER), которые могут иметь значения, но не имеют команд. К третьему типу относятся параметры-сборщики (COLLECTOR), которые имеют команды, но не имеют значений, не отображаются в консолях и, следовательно, параметрами называются достаточно условно.

Стандартный параметр — самый простой. Агент запускает соответствующую команду, а она измеряет и устанавливает значение параметра. Однако самый простой параметр является самым неэффективным, так как на каждый такой параметр необходимо запускать отдельный PSL-процесс. Учитывая также, что большинство реальных измерений требуют запуска минимум одного процесса операционной системы, то нагрузка на наблюдаемую систему может стать неоправданно большой.

С другой стороны, большинство применяемых в реальной жизни алгоритмов позволяют измерить сразу несколько характеристик одного или нескольких измеряемых объектов. Поэтому параметры-одиночки встречаются довольно редко, а измерения производятся при помощи одного или нескольких параметров-коллекторов, устанавливающих значения максимально возможного числа параметров-потребителей.

ЗАКЛЮЧЕНИЕ

К сожалению, формат статьи и дефицит времени не позволяют подробнее остановиться на тех или иных особенностях или охватить в рассказе больше инструментов. И все же хотелось бы надеяться, что заинтересованный читатель получил достаточно ясное представление о продукте. Желающие узнать более полную информацию на эту тему могут направлять свои вопросы автору по адресу: Jetinfo@jet.msk.su.

19 На самом деле все это можно делать и при помощи скриптов discovery, однако по ряду причин этот способ менее удобен.

Jet Info
ИНФОРМАЦИОННЫЙ БЮЛЛЕТЕНЬ
Издается с 1995 года
Издатель: компания «Инфосистемы Джет»

Главный редактор: Дмитриев В.Ю. *(vlad@jet.msk.su)*
Технический редактор: Лапина И.К. *(lapina@jet.msk.su)*
Россия, 127015, Москва, Б. Новодмитровская, 14/1
тел. (495) 411 76 01
факс (495) 411 76 02
email: JetInfo@jet.msk.su http://www.jetinfo.ru

Подписной индекс по каталогу Роспечати 32555