УДК 004.89:[004.042:004.416]

А.Ю. Крайнов, А.А. Смагин

АВТОМАТИЗАЦИЯ СБОРА И ОБРАБОТКИ ПРОТОКОЛОВ В СИСТЕМЕ СОПРОВОЖДЕНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ОСНОВЕ ОБРАБОТКИ СЛОЖНЫХ СОБЫТИЙ

Крайнов Александр Юрьевич, *аспирант кафедры «Телекоммуникационные технологии и сети» Ульяновского государственного университета, окончил факультет математики и информационных технологий УлГУ. Область научных интересов: интеллектуальный анализ данных, обработка текстов на естественном языке, архитектура программных средств. [e-mail: kralyu@mail.ru].*

Смагин Алексей Аркадьевич, *доктор технических наук, заведующий кафедрой «Телекоммуникационные технологии и сети» УлГУ, окончил радиотехнический факультет Ульяновского политехнического института. Имеет свыше 200 статей, изобретений, монографий в области разработки информационных систем различного значения. [e-mail: alsmagin@ulsu.ru].*

Аннотация

В статье описан подход к идентификации ошибок программного обеспечения (ПО), возникающих в процессе его выполнения. В основу подхода положен анализ последовательностей сообщений, сигнализирующих о проблемах в работе ПО, с помощью методов обработки сложных событий. На основе программного агента и системы обработки сложных событий Drools Fusion компании JBoss разработан макет системы сбора и классификации программных протоколов. Показано, что большинство ошибок может быть классифицировано с помощью автоматически сгенерированных правил. Разработанные средства обеспечивают перенастраиваемость алгоритмов реакции на возникающие ошибки.

Ключевые слова: сопровождение программного обеспечения, анализ программных протоколов, системы принятия решений, экспертные системы, системы управления бизнес-правилами, обработка потоков сообщений, обработка сложных событий, JBoss Drools.

Aleksandr Iurievich Krainov, *Post-graduate student at the Department of Telecommunications Technologies and Networks at Ulyanovsk State University; graduated from the Faculty of Mathematics and Information Technologies at Ulyanovsk State University; interested in intelligent data analysis, text processing in the natural language, and software architecture. e-mail: kralyu@mail.ru.*

Aleksei Arkadevich Smagin, *Doctor of Engineering, Head of the Department of Telecommunications Technologies and Networks at Ulyanovsk State University; graduated from the Faculty of Radio-Engineering of Ulyanovsk Polytechnic Institute; an author of more than 200 articles, inventions, and monographs in the field of different-purpose information system development. e-mail alsmagin@ulsu.ru.*

Abstract

The article presents an approach to software bug identification based on log stream analysis via complex event processing. The basis of the approach is the message sequences analysis indicating the software problems by means of the complex event processing methods. A layout system of collecting and classifying software protocols is designed on the basis of a software agent and a complex event processing system of JBoss Drools Fusion. The results prove that most errors can be classified via automatically generated rules. The developed tools provide a rebuilding of the reaction algorithms to the error emerging.

Key words: software maintenance, log analysis, decision-making systems, expert systems, business rules management systems, message flow processing, complex event processing, JBoss Drools

Введение

Сопровождение ПО – это сложный процесс, включающий большое количество стадий и затрагивающий множество участников. В цикле разработки ПО он оценивается как наиболее затратный в экономическом плане этап [1, p. 1181], доля которого варьируется от 50 до 80% [2, p. xii]. Повышение эффективности сопровождения ПО за счет автоматизации различных стадий этого процесса представляется в настоящее время актуальным.

Неотъемлемой частью процесса сопровождения является сбор и обработка данных от пользователей о найденных ими ошибках в работе ПО.

Важным показателем эффективности системы сбора сообщений об ошибках является время, прошедшее с момента поступления такого сообщения до момента принятия решения. В свою очередь, на данный показатель оказывает влияние множество других факторов:

1. Психологическое состояние лица, принимающего решения (ЛПР);

2. Правильность и непротиворечивость описания ошибки, данного пользователем;

3. Количество дубликатов ошибки, присланных разными пользователями;

4. Наличие прецедентов ошибки;

5. Критичность ошибки [3] и др.

Некоторые из вышеперечисленных факторов связаны с конкретными бизнес-процессами, действующими на предприятии-разработчике ПО. Эти процессы могут быть подвергнуты анализу и, в конечном итоге, автоматизированы. В настоящей работе предпринята попытка автоматизации поиска ошибок за счет анализа служебной информации, содержащейся в программных протоколах (ПП), которые генерируются ПО на стороне пользователя в процессе его выполнения.

Целью настоящей работы является сокращение времени обработки сообщений об ошибках в ПО. Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Исследование процесса обработки потока сообщений от пользователя;

2. Построение формальной модели системы обработки сообщений;

3. Идентификация ошибок в потоке сообщений с ПП;

4. Поиск аналогов ошибки в базе прецедентов;

5. Автоматизация процессов п. 3, п. 4.

1 Процесс обработки сообщений об ошибках от пользователя

Процесс обработки данных об ошибках в общем виде включает следующие шаги:

1. Сбор данных о работе ПО (ПП, потреблении ПО ресурсов рабочей станции);

2. Сбор пользовательских отчетов о проблемах в работе ПО;

3. Идентификация ошибок среди данных пп. 1, 2;

4. Создание агрегированного представления ошибки;

5. Исправление ошибки разработчиком;

6. Информирование пользователя о способах решения проблемы.

В зависимости от качества сопровождения, оказываемого разработчиком, некоторые шаги могут быть редуцированы или отсутствовать. Рассмотрим более внимательно шаги 1 и 2, представляющие собой процесс сбора данных об ошибке. Применительно к системе с использованием программного агента [4] этот процесс выглядит следующим образом.

Когда агент замечает возникновение исключительной ситуации, он начинает формирование сообщения о ней для передачи на сервер. При этом он запрашивает у пользователя описание действий, которые тот выполнял перед возникновением ошибки. Данный процесс повторяется для каждой новой ошибки, даже при условии, что описание подобной ошибки (прецедент) уже имеется в базе данных разработчика (например, если она уже возникала у других пользователей и те дали исчерпывающую информацию о проблеме). Агент также не имеет возможности проинформировать пользователя о путях решения проблемы в том случае, когда возникшая ошибка уже была исправлена разработчиком (например, если она решается установкой пакета обновления).

Чтобы иметь возможность перенастраивать поведение программного агента в процессе его работы, необходимо дополнить систему обработки ошибок подсистемой анализа протоколов, которая будет осуществлять поиск прецедента в базе знаний разработчика [5].

При возникновении ошибочной ситуации агент формирует не полное сообщение, а лишь минимально необходимую запись о происшедшем событии. Полное сообщение об ошибке (на основании отчета пользователя) создается только тогда, когда подсистема анализа протоколов не находит прецедента. Напротив, если прецедент был найден и известны способы решения возникшей проблемы, пользователю о них сообщается. Соответствующий бизнес-процесс будет выглядеть следующим образом (рис. 1).

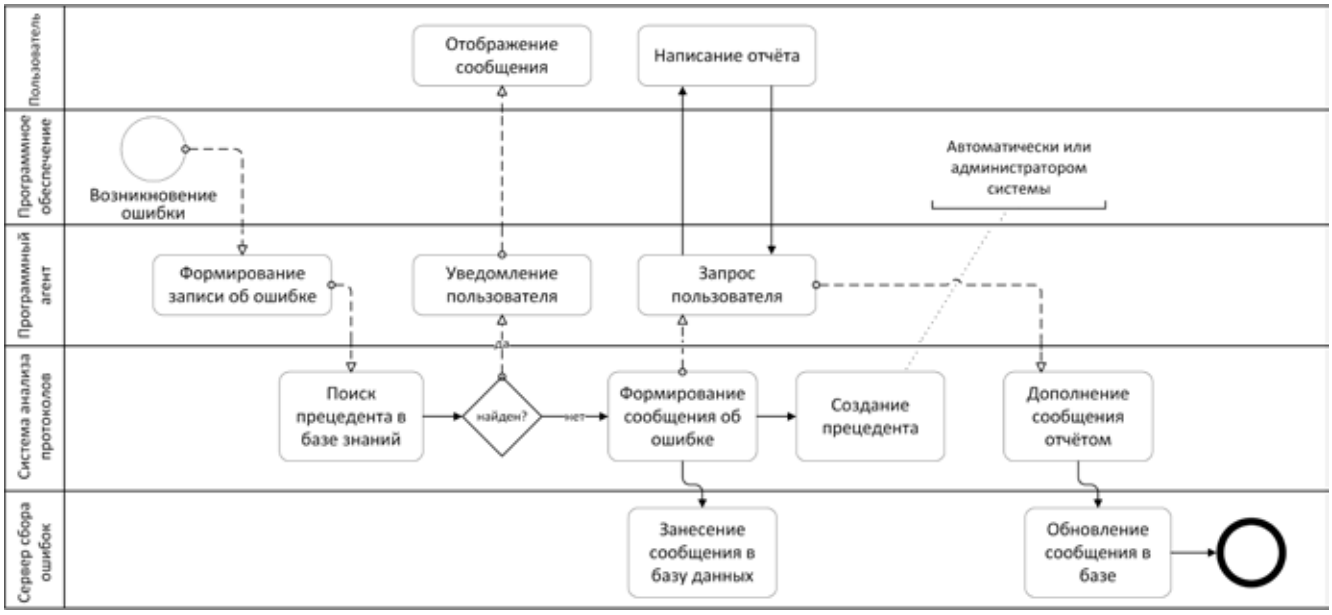


Рис. 1. Процесс сбора данных об ошибках с помощью агента при использовании подсистемы анализа прецедентов

2 Формальная модель подсистемы анализа протоколов в сообщениях

Будем считать *событием* (E) зарегистрированное изменение состояния ПО или параметров рабочей станции пользователя в процессе выполнения ПО. Идентификация событий на стороне разработчика производится на основе анализа сообщений с ПП.

Ошибка (B) – последовательность событий, которая указывает на отклонения процесса выполнения ПО от заданного. Ошибка может сопровождаться пользовательским комментарием (описанием ошибочной ситуации на естественном языке).

Рассмотрим пример. Пусть E_1 – зарегистрированное в протоколе событие чтения некоторого файла (рис. 2), причем вследствие неточного указания имени файла в процессе выполнения ПО возникает первоначальная ошибка. Пусть имеются также E_2, E_3, E_6 – шумовые события, то есть записи, занесенные в ПП после E_1, но не имеющие отношения к нарушению процесса выполнения ПО. Событие E_4 – факт попытки обращения к файлу. В событии E_5 может оказаться, что данные из файла не могут быть считаны. В событии E_7 фиксируется возникновение критической ошибки в работе ПО.

Если E_1 – источник первоначальной ошибки, а E_4, E_5, E_7 – события, в которых происходит ухудшение результатов процесса выполнения ПО вследствие первоначальной ошибки, то мы можем говорить, что последовательность (E_1, E_4, E_5, E_7) образует ошибку B_1.

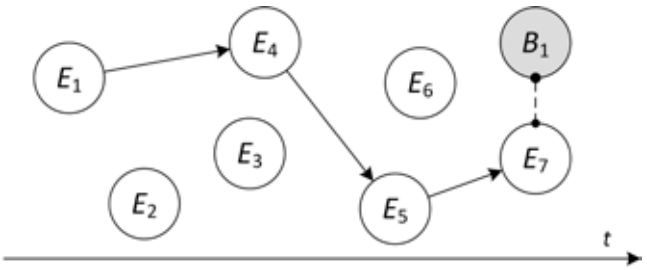


Рис. 2. Пример последовательности событий, образующих ошибку

Контекстную модель подсистемы анализа протоколов (см. рис. 1) можно представить в виде «черного ящика» (рис. 3).

Рис. 3. Подсистема анализа протоколов в виде «черного ящика»

Модель системы в общем виде имеет вид:

$$Y = \Theta(X).$$

Входом X системы являются сообщения M с протоколами от программных агентов. На M накладываются следующие ограничения:

t – дата и время генерации сообщения источником (выражается положительным целым числом);

s – критичность события (число в интервале $[1;Z]$, где Z – количество уровней критичности для источника сообщений);

c – наименование подпрограммы ПО, сгенерировавшей событие (используется при классификации событий, выражается строкой символов);

m – автоматически сгенерированный текст сообщения (используется при классификации событий, выражается строкой символов).

Выход Y – это пара (B, P), где B – ошибка, а P – прецедент этой ошибки. Для описания ограничений на эти переменные введем дополнительные формальные определения.

Событие E_M, образованное на основе сообщения M, представляется в виде множества признаков:

$$E_M = (t, s, \overline{c}, \overline{m}),$$

где $t \in \mathbb{R}$ и $s \in \mathbb{R}$ – время генерации и критичность сообщения M;

c – множество подстрок строки c (представляющей подпрограмму-источник сообщения M);

m – множество подстрок строки m (представляющей текст сообщения M).

Обозначим через $\mathbb{E}$ множество всех зарегистрированных разработчиком событий:

$$\mathbb{E} = (E_1, E_2, ..., E_n).$$

При поступлении в момент времени t сообщения M_t соответствующее этому сообщению событие добавляется к общему множеству событий:

$$\mathbb{E}_t = \mathbb{E}_{t-1} \cup E_{M_t}.$$

В соответствии с данным выше определением, ошибка B представляется в виде упорядоченного по времени подмножества событий из $\mathbb{E}$:

$$B = \left(E_{j_1}(t_1), E_{j_2}(t_2), \ldots, E_{j_k}(t_k)\right), j \in \mathbb{R},$$
$$j_p > j_q \rightarrow t_p > t_q.$$

Для дальнейших рассуждений удобно, чтобы ошибка не зависела от конечного момента времени $t_k = t_B$ (времени появления последнего события E_{jk} или времени возникновения ошибки B). Для этого определим *прецеденты событий в составе ошибки* (F) следующим образом:

$$F_i = \left(\Delta t_{F_i}, s_{F_i}, \overline{c_{F_i}}, \overline{m_{F_i}}\right),$$
$$\Delta t_{F_i} = t_B - t_{E_i}, \overline{c_{F_i}} \subset \overline{c_{E_i}}, \overline{m_{F_i}} \subset \overline{m_{E_i}},$$

где Δt_{F_i} – интервал времени от события E_i до ошибки B. Множества $\overline{c_F}$ и $\overline{m_F}$ прецедента события F являются собственными подмножествами $\overline{c_E}$ и $\overline{m_E}$ события E, что позволяет обеспечить соответствие прецеденту нескольких событий.

На основании последовательности прецедентов событий можно определить прецеденты для ошибок P:

$$P = (p, F_1, F_2, ..., F_k),$$

где $p \in \mathbb{Z}$ – приоритет прецедента ошибки над другими прецедентами, задаваемый разработчиком.

Преобразование Θ должно обнаруживать и идентифицировать ошибки в последовательностях событий на основании анализа выделенной атрибутики сообщений M и выдавать рекомендации по устранению тех ситуаций, в которых ошибка может возникнуть.

Выразить Θ математической формулой, графиком или матричным произведением не представляется возможным, т. к. это преобразование является сложным и включает в себя идентификацию событий, ошибок, а также процедуры поиска. Поэтому в качестве Θ выступает алгоритм.

Требования к алгоритму Θ:

- обеспечение точного преобразования X в Y;
- сохранение промежуточных результатов преобразований;
- работа с базами прецедентов;
- быстрый поиск аналогов ошибки в базе.

Опишем общую структуру алгоритма. С этой целью декомпозируем его контекстную модель (рис. 4).

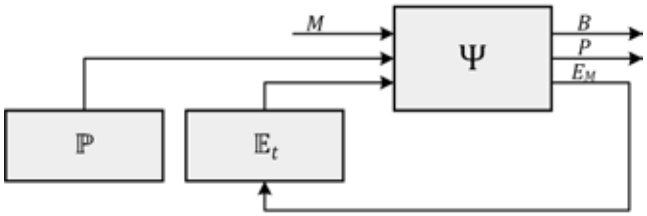


Рис. 4. Декомпозиция контекстной модели подсистемы анализа прецедентов

Декомпозиция позволяет выделить для дальнейшего анализа две базы: базу $\mathbb{P}$ прецедентов и базу $\mathbb{E}_t$ зарегистрированных на момент t поступления сообщения M событий – а также преобразование Ψ, представляющее собой подсистему поиска прецедентов. На основании декомпозиции опишем алгоритм Θ в общем виде.

Алгоритм Θ можно представить в виде следующей последовательности шагов:

1. Извлечь событие E_t из сообщения M_t, поступившего в момент времени t;
2. Добавить событие E_t в базу $\mathbb{E}_t$ всех событий, зарегистрированных на момент времени t;
3. Построить прецедент события F_E;
4. Выбрать из базы $\mathbb{P}$ прецеденты ошибок $\mathbb{P}_E$, имеющие в своем составе прецедент события F_E;
5. Выбрать из $\mathbb{P}_E$ прецеденты $\mathbb{P}_{E,t}$, соответствующие множеству $\mathbb{E}_t$ зарегистрированных на момент времени t событий;
6. Выбрать из $\mathbb{P}_{E,t}$ прецедент ошибки P с максимальным значением приоритета p;
7. Найти события $\mathbb{P}_{P,t}$, соответствующие прецеденту P, и на их основе построить ошибку B.

В алгоритме соответствие между прецедентами и событиями устанавливается на основе четырех рассмотренных выше параметров событий:

- событие E произошло в интервале прецедента события F с учетом коэффициента погрешности времени $\tau \in \mathbb{R}$;
- уровни критичности ошибки и прецедента равны;
- все подстроки $\overline{c}$ у прецедента входят в соответствующее множество у события;
- все подстроки $\overline{m}$ у прецедента входят в соответствующее множество у события.

Иными словами, прецедент P соответствует множеству зарегистрированных на момент времени t событий при выполнении следующего условия:

$$\forall F_i \in P \ \exists E_j \in \mathbb{E}_t : \left(\Delta t_{\mathrm{F}_i} \geq \frac{1}{\tau}\left(t_{F_{max}} - t_{\mathrm{E}_j}\right)\right) \wedge$$

$$\wedge \left(s_{F_i} = s_{E_j}\right) \wedge \left(\overline{c_{F_i}} \subset \overline{c_{E_j}}\right) \wedge \left(\overline{m_{F_i}} \subset \overline{m_{E_j}}\right).$$

Заметим, что одной ошибке может соответствовать более одного прецедента, а одному прецеденту – более одной ошибки.

3 Разработка системы обработки сообщений

3.1 Выбор средств реализации системы обработки сообщений

Среди систем управления бизнес-процессами (business process management systems, BPMS) возможностями учета параметров времени обладают системы обработки информационных потоков (information flow processing, IFP [6]). Наиболее общим подходом в рамках IFP является анализ облаков событий (event clouds) – множеств неупорядоченных сообщений, значимость информации которых не определена. В основу этого подхода положены методы обработки сложных событий (complex event processing, CEP), в свою очередь, базирующихся на методах временных логик.

Для реализации предложенного подхода выбрана система обработки сложных событий Drools Fusion [7, с. 133; 8, с. 119; 9, с. 283]. Выбор именно этой системы обусловлен возможностью интеграции разрабатываемой системы с другими компонентами Drools, в том числе машиной исполнения правил (rules engine) – для обработки структурированных параметров записей.

Авторами разработан макет системы обработки сообщений с использованием следующих средств и технологий:

- среды разработки Eclipse Java EE Juno SR2;
- виртуальной машины Java 1.7 (HotSpot);
- библиотеки интерфейса Swing GUI (Nimbus);
- библиотеки протоколирования slf4j 1.7.2;
- библиотеки протоколирования log4j 1.2.17;
- системы гарантированной доставки сообщений JBoss HornetQ 2.3.0.CR1;

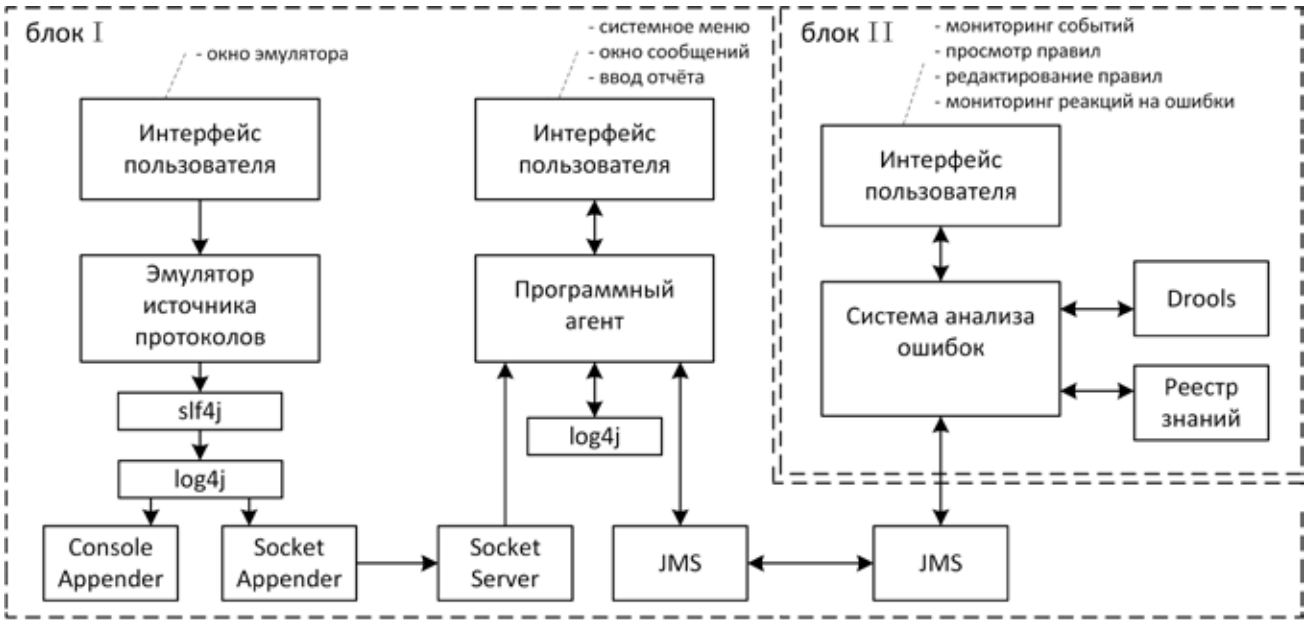


Рис. 5. Структура макета системы анализа ошибок

- машины исполнения правил JBoss Drools 5.5.0.Final (компоненты Expert и Fusion).

Макет построен на основе декомпозированной контекстной модели (рис. 4) и включает два основных блока: блок I – компоненты, обеспечивающие генерацию и сбор протоколов; блок II – собственно подсистема анализа протоколов (рис. 5).

3.2 Сбор и подготовка данных

Генерация тестовых программных протоколов и ошибок производится с помощью специального эмулятора с графическим интерфейсом, способного создавать как шумовые случайные последовательности записей протоколов, так и заранее определенные цепочки из 2-х или 3-х записей. Так как прямое взаимодействие источника протоколов и программного агента невозможно из-за принципиальных ограничений виртуальной машины Java, для сбора протоколов используется адаптер SocketAppender компонента log4j, который направляет копии протоколов на локальный для тестового ПК порт № 4714.

Этот порт, в свою очередь, прослушивается сервером сокетов, который регистрируется программным агентом. Поступающие сообщения (соответствующие событиям E) обрабатываются с помощью вспомогательных средств log4j, после чего из них извлекаются следующие данные:

- время генерации сообщения источником t_E (число в миллисекундах);
- степень критичности сообщения s_E (число от 1 до 5);
- класс Java c_E, сгенерировавший сообщение (строка);
- текст сообщения m_E (строка).

Полученные данные кодируются в сообщение в формате XML, которое записывается в очередь JMS.

Система анализа ошибок считывает сообщения из очереди XML и преобразовывает их в объект-событие, пригодный к обработке в системе (тем самым выполняется шаг 1 алгоритма, разработанного в п. 2 статьи). В дополнение к вышеперечисленным данным, он хранит идентификатор и информацию о ходе своей обработки. Сформированный объект передается в подсистему пользовательского интерфейса (рис. 6) и подсистему исполнения правил, которая помещает его в рабочую память Drools Fusion (шаг 2 алгоритма).

3.3 Применение правил классификации и формирование сообщений об ошибках

При получении нового события машина Drools Fusion сопоставляет все хранимые в своей рабочей памяти события имеющимся в базе знаний правилам (описывающим прецеденты ошибок $\mathbb{P}$).

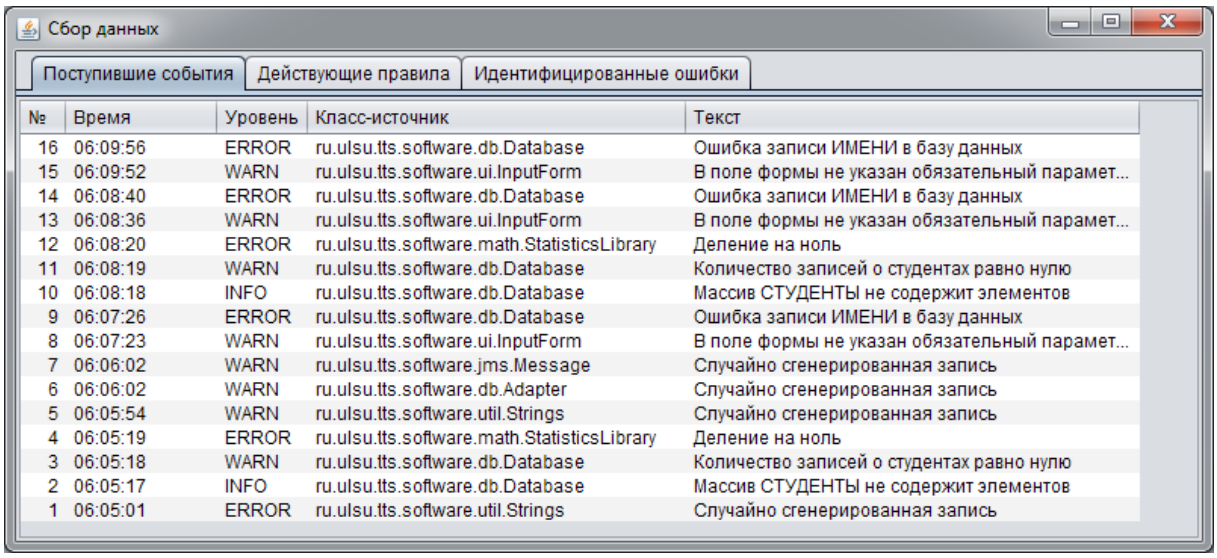
В качестве примера рассмотрим события №№ 2, 3, 4 в таблице на рисунке 6. Они представляют ситуацию, в которой по причине отсутствия проверки на нулевое количество элементов в списке студентов (о последнем свидетельствует запись протокола «WARN»), загружаемого из базы данных (о чем говорит имя класса «Database» в записи уровня «INFO»), возникает ошибка деления на ноль уровня «ERROR».

Как было показано в п. 2, ошибке, состоящей из этих трех событий, может соответствовать несколько прецедентов. Одним из возможных вариантов будет следующий перечень одновременных условий:

- возникло событие уровня 5 («ERROR»), содержащее в тексте сообщения строку «Деление на ноль», и
- перед ним (за 15 минут до него) возникло событие уровня 4 («WARN»), содержащее в тексте сообщения строку «равно нулю», и
- перед ним (за 2 секунды до него) возникло событие, содержащее в имени класса-источника строку «Database».

Эти условия могут быть формализованы в виде следующего выражения:

$$\left(\left(5 = s_{E_3}\right) \wedge \left(\text{«}Деление\ на\ ноль\text{»} \subset \overline{m_{E_3}}\right)\right) \wedge$$
$$\left(\left(2000 \geq \frac{1}{2}\left(t_B - t_{E_1}\right)\right) \wedge \left(\text{«Database»} \subset \overline{c_{E_1}}\right)\right).$$



№	Время	Уровень	Класс-источник	Текст
16	06:09:56	ERROR	ru.ulsu.tts.software.db.Database	Ошибка записи ИМЕНИ в базу данных
15	06:09:52	WARN	ru.ulsu.tts.software.ui.InputForm	В поле формы не указан обязательный парамет...
14	06:08:40	ERROR	ru.ulsu.tts.software.db.Database	Ошибка записи ИМЕНИ в базу данных
13	06:08:36	WARN	ru.ulsu.tts.software.ui.InputForm	В поле формы не указан обязательный парамет...
12	06:08:20	ERROR	ru.ulsu.tts.software.math.StatisticsLibrary	Деление на ноль
11	06:08:19	WARN	ru.ulsu.tts.software.db.Database	Количество записей о студентах равно нулю
10	06:08:18	INFO	ru.ulsu.tts.software.db.Database	Массив СТУДЕНТЫ не содержит элементов
9	06:07:26	ERROR	ru.ulsu.tts.software.db.Database	Ошибка записи ИМЕНИ в базу данных
8	06:07:23	WARN	ru.ulsu.tts.software.ui.InputForm	В поле формы не указан обязательный парамет...
7	06:06:02	WARN	ru.ulsu.tts.software.jms.Message	Случайно сгенерированная запись
6	06:06:02	WARN	ru.ulsu.tts.software.db.Adapter	Случайно сгенерированная запись
5	06:05:54	WARN	ru.ulsu.tts.software.util.Strings	Случайно сгенерированная запись
4	06:05:19	ERROR	ru.ulsu.tts.software.math.StatisticsLibrary	Деление на ноль
3	06:05:18	WARN	ru.ulsu.tts.software.db.Database	Количество записей о студентах равно нулю
2	06:05:17	INFO	ru.ulsu.tts.software.db.Database	Массив СТУДЕНТЫ не содержит элементов
1	06:05:01	ERROR	ru.ulsu.tts.software.util.Strings	Случайно сгенерированная запись

Рис. 6. Интерфейс администратора. Список поступивших записей о событиях

Соответствующее правило, выраженное на внутреннем для системы JBoss Drools языке описания DRL, приведено в листинге 1.

В строках 2–4 объявлены внешние для машины Drools классы созданной системы, которые будут использоваться в процессе исполнения правил. В строках 5–9 объявляется внутренний объект рабочей памяти, который будет обрабатываться в системе. В строке 8 указывается, что по истечении 30 минут объект должен быть удален из рабочей памяти.

В строках 11–20 определяется правило реагирования на последовательность событий. Строка 12 запрещает активацию других правил после внесения данным правилом изменений в обрабатываемый объект. Строка 13 устанавливает приоритет p прецедента, соотнесенного с правилом, по отношению к другим прецедентам.

Строки 15, 16 и 17 задают условия правила (антецедент), соответствующие прецедентам событий F. При проверке этих условий на соответствие поступившим сообщениям машина Drools Fusion осуществляет построение прецедентов событий F (шаг 3 алгоритма из п. 2 статьи). Одновременное выполнение всех условий правила приводит к его активизации, что соответствует выбору прецедента, ассоциированного с правилом (шаги 4, 5 алгоритма). Благодаря параметрам правила, определенным в строках 12–13, гарантируется, что среди всех подходящих прецедентов будет выбран прецедент с наивысшим приоритетом (шаг 6 алгоритма).

При активации правила найденные события, совпадающие с прецедентами, сохраняются в переменных $error, $PP_CAUSE_1 и $PP_CAUSE_2. В строках 19–21 на основании этих событий создается запись об ошибке B (шаг 7 алгоритма).

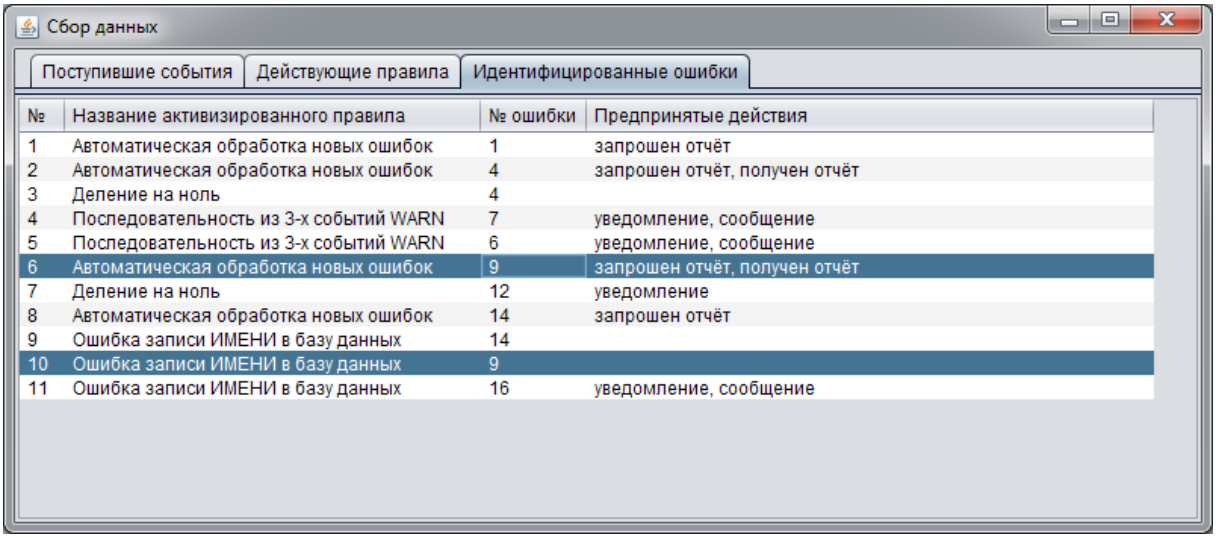
Листинг 1

Правило «Деление на ноль» на языке DRL

```
1  package iam.ilog.server.rules
2  import iam.ilog.server.domain.LogEntryEvent;
3  import iam.ilog.server.domain.ComplexErrorEvent;
4  import iam.ilog.server.EventFactory;

5  declare LogEntryEvent
6      @role(event)
7      @timestamp(time)
8      @expires( 30m )
9  end

11 rule "Деление на ноль"
12   lock-on-active
13   salience 0
14 when
15   $error : LogEntryEvent ( level == 5,
         this.messageMatches("Деление на ноль") ) from entry-point "MainStream"
16   $PP_CAUSE_1 : LogEntryEvent ( level == 4, this.messageMatches("равно
нулю"),
17       this before[ 1ms, 30m ] $error ) from entry-point "MainStream"
     $PP_CAUSE_2 : LogEntryEvent ( this before[ 1ms, 4s ] $error,
18       this.classMatches("Database") ) from entry-point "MainStream"
19 then
     ComplexErrorEvent $complexEvent
20       = new ComplexErrorEvent( $error, drools.getRule().getName() );
21   EventFactory.addCauseToComplexEvent( $complexEvent, $PP_CAUSE_1 );
22   EventFactory.addCauseToComplexEvent( $complexEvent, $PP_CAUSE_2 );
23   EventFactory.doShowNotification( $complexEvent, "Возникла ошибка" );
24   EventFactory.doAskReport( $complexEvent, "Напишите отчет:" );
25   EventFactory.fireComplexEvent( $complexEvent );
26   modify ($error) { setProcessed(true) }
   end
```



№	Название активизированного правила	№ ошибки	Предпринятые действия
1	Автоматическая обработка новых ошибок	1	запрошен отчёт
2	Автоматическая обработка новых ошибок	4	запрошен отчёт, получен отчёт
3	Деление на ноль	4	
4	Последовательность из 3-х событий WARN	7	уведомление, сообщение
5	Последовательность из 3-х событий WARN	6	уведомление, сообщение
6	Автоматическая обработка новых ошибок	9	запрошен отчёт, получен отчёт
7	Деление на ноль	12	уведомление
8	Автоматическая обработка новых ошибок	14	запрошен отчёт
9	Ошибка записи ИМЕНИ в базу данных	14	
10	Ошибка записи ИМЕНИ в базу данных	9	
11	Ошибка записи ИМЕНИ в базу данных	16	уведомление, сообщение

Рис. 7. Интерфейс администратора. Список идентифицированных ошибок

Строки 22—23 определяют нужную реакцию программного агента: в данном случае здесь вызываются методы, которые формируют и отсылают агенту уведомление и запрос на дополнительную информацию от пользователя. Строка 24 вызывает метод, который завершает создание сообщения об ошибке и записывает его в базу данных. Строка 25 переводит сообщение в состояние «обработано», чтобы оно не приводило к активации других правил.

3.4 Результаты работы системы

Разработанный макет предусматривает следующие варианты реакций на входящую запись об ошибке:

- игнорирование;
- создание сообщения об ошибке и сохранение ее в базе данных;
- показ пользователю клиентской машины всплывающего сообщения через область системных уведомлений (трей);
- показ пользователю сообщения через стандартное диалоговое окно.
- запрос у пользователя описания действий, которые могли спровоцировать возникновение ошибки.

Текущий набор реакций не является исчерпывающим, при необходимости поведение системы может быть расширено путем модификации специального программного класса.

Администратор имеет возможность просмотреть список сформированных на основании записей журналов сообщений об ошибках и предпринятых системой действиях (рис. 7).

Заключение

Для решения задачи автоматизации сбора протоколов в процессе сопровождения ПО разработаны модель и программная система на основе процедуры обработки сложных событий, что позволило учитывать временные параметры ошибок, а также предоставить возможность перенастраивать алгоритмы реакции программного агента.

Разработанный макет системы, состоящий из программного агента, системы управления бизнес-правилами и базы прецедентов ошибок, позволяет осуществлять сбор ПП, классифицировать новые ошибки и управлять обратной связью с пользователем. На практике это приводит к повышению эффективности и качества обработки получаемых сообщений. Испытания макета подтвердили корректность предложенного подхода.

СПИСОК ЛИТЕРАТУРЫ

1. Rodríguez O.M. etal. Using a Multi-agent Architecture to Manage Knowledge in the Software Maintenance Process // Knowledge-Based Intelligent Information and Engineering Systems / Ed. by Negoita M.G., Howlett R.J., Jain L.C. Springer Berlin Heidelberg, 2004. pp. 1181–1188.

2. Jarzabek S. Effective Software Maintenance and Evolution: A Reuse-Based Approach. Auerbach Publications, 2007. 424 p.

3. Дроботун Е.Б. Критичность ошибок в программном обеспечении и анализ их последствий // Фундаментальные исследования. – 2009. – Т. 4. – С. 73–74.

4. Построение системы доставки обновлений программных продуктов /Захаров В.Г. [и др.] // Ученые записки Ульяновского государственного университета / под ред. проф. А.А. Смагина. – 2012. – № 1 (4). – С. 161–174.

5. Крайнов А.Ю. Применение систем принятия решений в системах сопровождения программной продукции // Ученые записки Ульяновского государственного университета / под ред. проф. А.А. Смагина. – 2012. – № 1 (4). – С. 194–200.

6. Cugola G., Margara A. Processing flows of information: From data stream to complex event processing // ACM Comput. Surv. 2012. Vol. 44, № 3. P. 15:1–15:62.

7. Bali M. Drools JBoss Rules 5.0 Developer's Guide. Packt Publishing, 2009. 320 p.

8. Amador L. Drools Developer's Cookbook. Packt Publishing, 2012. 310 p.

9. Salatino M., Aliverti E. jBPM5 Developer Guide. Packt Publishing, 2012. 364 p.