Автоматизация поиска ошибок в системах диспетчерского контроля железных дорог

## **Диспетчерский контроль на железных дорогах России**

На железных дорогах используется огромное количество систем безопасности движения транспорта, к которым зачастую нет физического доступа. Так как ручной поиск ошибок и неисправностей в этих системах невозможен, используется системы удаленного мониторинга, содержащие базы данных по всем устройствам системы. Базы данных формируются человеком и могут содержать неправильно введенные данные. Поскольку базы слишком велики, необходимо автоматизировать поиск возможных ошибок. Рассмотрим подробней системы удалённого мониторинга.

Для мониторинга состояния устройств железнодорожной автоматики и телемеханики (ЖАТ) на железных дорогах России используются аппаратно-программные комплексы диспетчерского контроля (системы мониторинга).

## **Проблема диспетчерского контроля**

После подключения устройства ЖАТ к системе мониторинга, в базе данных появляется информация о подключенном устройстве. В базу записываются все события о неисправности, связанные с этим объектом.

Чтобы отследить неисправность устройства ЖАТ, инженер должен привязать в базе данных необходимую индикацию к соответствующему событию объекта.

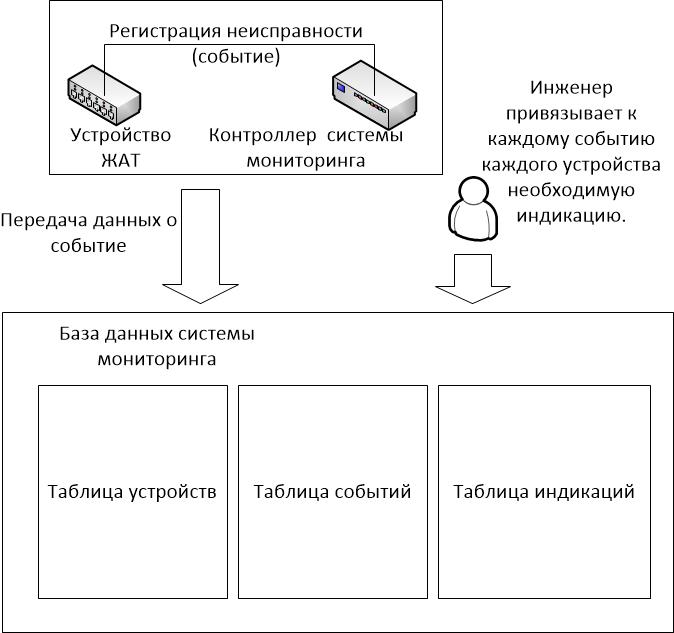
****

Рисунок 1. Схема привязок.

Например, инженер может привязать к событию «Падение напряжения» индикацию «мигание красным». После того, как на определенном устройстве упадет напряжение, пользователи увидят индикацию, при которой соответствующее устройство на пульте управления мигает красным цветом.

Так как количество объектов и возможных событий очень велико, одной из главных проблем является ручная привязка событий к объектам и индикациям. Инженер может не привязать нужное событие к индикации, привязать не к тому объекту или не к той индикации. Таким образом, возвращаясь к нашему примеру, пользователь ПО не увидит, что на устройстве упало напряжение.

## **Автоматизация поиска ошибок**

Для решения проблемы была разработана программа, которая сканирует базу данных на среднем уровне и выявляет возможные ошибки на основании указанных правил.

**Входные данные**: На вход в программу подаются база данных и правила поиска ошибок, описанные в виде кода на языке С#.

**Выходные данные:** Результатом работы программы являются таблицы с информацией о кол-ве ошибок и детализацией каждой ошибки.

**База данных**

Для работы алгоритм использует следующие таблицы из базы данных:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Таблица объектов** | | | | |
| **Столбец** | id | type | group\_id | name |
| **Тип данных** | int | int | int | string |
| **Комментарий** | Идентификатор объекта | Тип объекта | Идентификатор группы объекта | Выводимое текстовое имя объекта |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Таблица событий** | | | | |
| **Столбец** | id | type | object\_id | name |
| **Тип данных** | int | int | int | string |
| **Комментарий** | Идентификатор события | Тип события | Идентификатор объекта, к которому при­вязано событие | Выводимое текстовое имя события |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Таблица индикаций** | | | | |
| **Столбец** | id | type | event\_id | name |
| **Тип данных** | int | int | int | string |
| **Комментарий** | Идентификатор индикации | Тип индика-ции | Идентификатор события, к которому привязана индикация | Выводимое текстовое имя индикации |

**Схема работы программы**

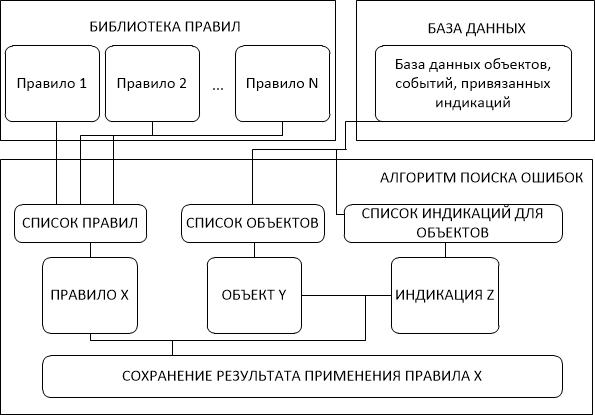
****

Рисунок 2.Схема работы алгоритма поиска ошибок.

Алгоритм загружает список правил из библиотеки правил, список объектов и привязанных индикаций из таблиц в базе данных. Затем последовательно проверяет соблюдается ли правило Х для индикаций Z объекта Y, и ,в случае нарушения правила, сохраняет информацию о возможной ошибке. В результате, алгоритм группирует ошибки по **станциям (области действия алгоритма)** и выводит пользователю.

**Результаты выводятся следующим образом:**

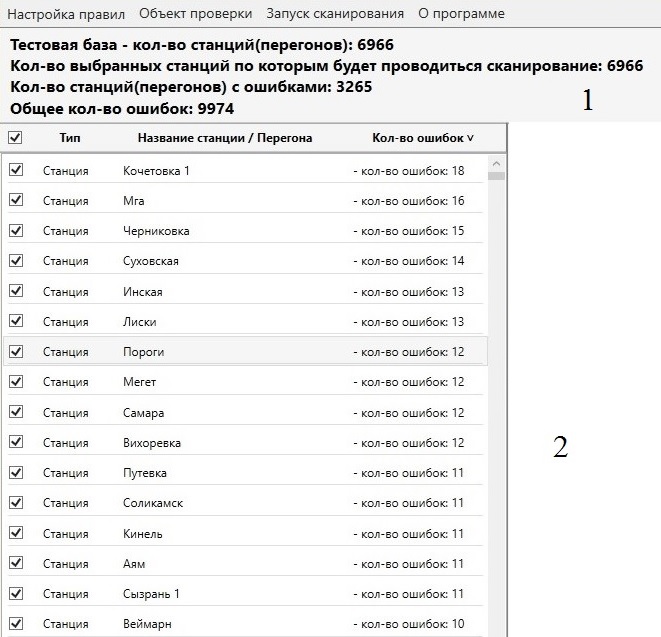


Рисунок 3 Окно вывода результата работы алгоритма.

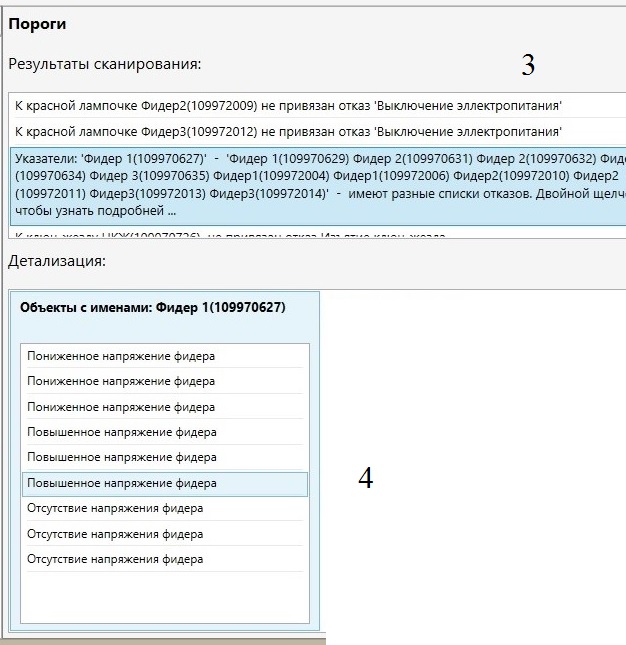


Рисунок 4. Окно вывода результата работы алгоритма.

*1 – Вывод общей информации о кол-ве просканированных станций и кол-ве найденных ошибок.*

*2 – Вывод информации о названии станции и кол-ве найденных ошибок.*

*3 – Вывод информации о найденных ошибках для выделенной станции.*

*4 – Вывод детальной информации о выделенной ошибке.*

**Правила поиска ошибок**

Правила поиска объединены в библиотеку правил. Каждое правило имеет текстовое описание и алгоритм работы. Все правила реализуют общие интерфейсы и способны без дополнительного кода встраиваться в библиотеку и исполняемый код.

**Пример правила:**

*// Сканирование светофоров  
// 1. Все объекты с типом 2 (светофоры) и именем '1' или '2' (предвходные) должны иметь одинаковый список отказов.  
// 2. Все остальные светофоры должны иметь одинаковый список отказов.*

foreach (DataRow traffic in dt\_objects.Select("ids =" + stance.id + " and type = 2"))  
{  
 if ((string)traffic["name"] == "2" || (string)traffic["name"] == "1")   
{  
enters.Add(get\_situation(traffic));   
}else  
{  
lights.Add(get\_situation(traffic));  
}}

**Выводы**

Применение автоматизации поиска ошибок на реальных дорогах дало следующие результаты:

**Результаты запуска алгоритма для 5 правил:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Кол-во станций | 6996 | 4010 | 2110 | 800 | 400 |
| Кол-во объектов | 9094800 | 5213000 | 2743000 | 1040000 | 520000 |
| Кол-во привязок | 63663600 | 36491000 | 19201000 | 7280000 | 3640000 |
| Кол-во вероятных ошибок | 3265 | 1265 | 982 | 300 | 112 |
| Процент правильно найденных ошибок | 15 | 20 | 20 | 40 | 45 |

Полученные результаты выявили следующие проблемы:

1. Процент правильно найденных ошибок увеличивается с уменьшением кол-ва сканируемых станций (область действия алгоритма). Проблема связана с тем, что к разным группам станций нужно писать разные правила и чем меньше групп мы сканируем, тем больше процент корректности результатов.
2. Кол-во найденных ошибок слишком мало и на практике мы смогли покрыть только 5-10% от общего числа содержащихся ошибок.

Исследование причин неточности алгоритма говорит о том, что вручную написанные правила поиска ошибок не дают необходимой эффективности.

Существует два варианта решения этой проблемы:

1. Постоянное пополнение, изменение и расширение библиотеки правил, что является трудоемкой ручной работой и, в конечном счете, также может содержать ошибки.
2. Применение методов интеллектуального анализа данных для автоматического построения библиотеки правил.

**Применение методов интеллектуального анализа данных**

Для поиска ошибок в базе данных все объекты необходимо разделить на группы согласно основным признакам (тип, имя, расположение объекта и т.д. // признаки надо уточнить), т.е. классифицировать. Значения признаков не заданы заранее.

Для каждого класса объектов найти наиболее часто встречающийся набор характеристик и принять эту последовательность за шаблон класса.

Каждый объект в базе должен соответствовать шаблону своего класса.

**Алгоритм поиска ошибок**

1. Классифицировать объекты.
2. Для каждого класса определить шаблон.
3. Для каждого объекта пометить характеристики, не соответствующие шаблону класса, как вероятные ошибки.

**Основные признаки объекта**

**Имя –** текстовое значение, кратко описывающее объект. Может быть любым, но обычно, в рамках класса, соответствует некоторому регулярному выражению. Например, (//здесь приведу конкретные примеры с базы).

**Тип объекта** –числовое значение, соответствующее одному из заданных устройств. Например, Тип 7 – светофор, Тип 8 – стрелка и т.д.

Только по типу классифицировать объект нельзя, потому что устройства нужно классифицировать по назначению (для этого нужно имя и расположение).

**Расположение объекта** - числовое значение, соответствующее одному из возможных расположений объекта. Объекты могут располагаться независимо друг от друга, тогда числовое значение расположения равно 0. Если два объекта имеют одинаковое значение, то они располагаются в одной области.

**Характеристики объекта**

Характеристиками объекта в нашем случае являются неисправности и индикации неисправностей, подключенных к данному объекту.

**Неисправности -** числовые значения, соответствующие возможным неисправностям на объекте. Например: неисправность 10 – «отключение питания», неисправность 12 – «короткое замыкание».

Каждый объект может иметь любое кол-во неисправностей. Добавление (т.е. привязка) неисправностей к объекту в базе происходит вручную.

**Индикация неисправностей -** числовое значение, соответствующее коду индикации для неисправности, определяет отображение заданной неисправности объекта на пульте диспетчера.

Каждая неисправность имеет свою индикацию. Во время добавления (привязки) неисправностей для объекта происходит и выбор индикации. Данный процесс не автоматизирован.

**Шаблон класса**

У объекта каждого класса должны быть одни и те же неисправности и индикации неисправностей, т.е. определённый набор характеристик. Наиболее часто встречающийся набор таких характеристик (например, у 90% объектов класса) определяется шаблоном объекта. Набор характеристик объектов, который не соответствуют шаблону, считается вероятно ошибочным.

**Задачей алгоритма** является поиск вероятных ошибок в базе объектов, с максимальным значением правильных предположений. Окончательное решение об исправлениях будет принимать человек.