## **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). Они состоят из трех уровней:

1. Нижний уровень - первичная обработка информации, снимаемой с устройств ЖАТ.

2. Средний уровень - сбор информации от подсистемы нижнего уровня, обработка, хранение, архивация и передача информации на верхний уровень.

3. Верхний уровень - обработка и вывод информации о неисправности устройств ЖАТ.

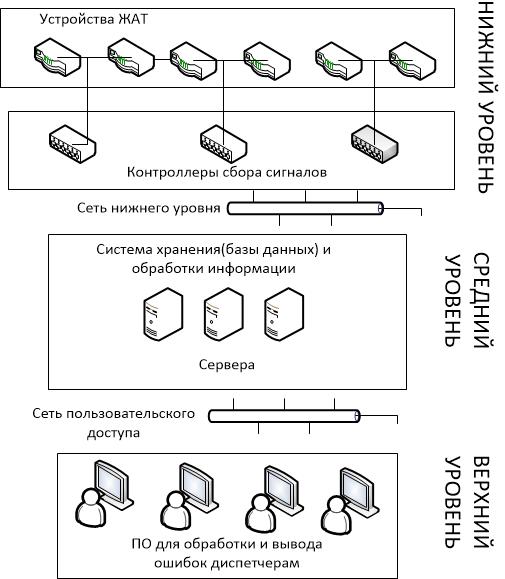


Рисунок 1.Схема организации уровней системы мониторинга

## **Проблема диспетчерского контроля**

После подключения устройства ЖАТ к системе мониторинга, на среднем уровне системы (в базе данных) появляются данные о подключенном устройстве(объекте). В базу записываются все события о неисправности, связанные с этим объектом.

Чтобы отследить неисправность устройства ЖАТ, инженер должен привязать к соответствующему событию объекта (короткое замыкание, падение напряжения и т.п.) в базе данных необходимую индикацию в пользовательском ПО (программа, работающая на верхнем уровне системы мониторинга).

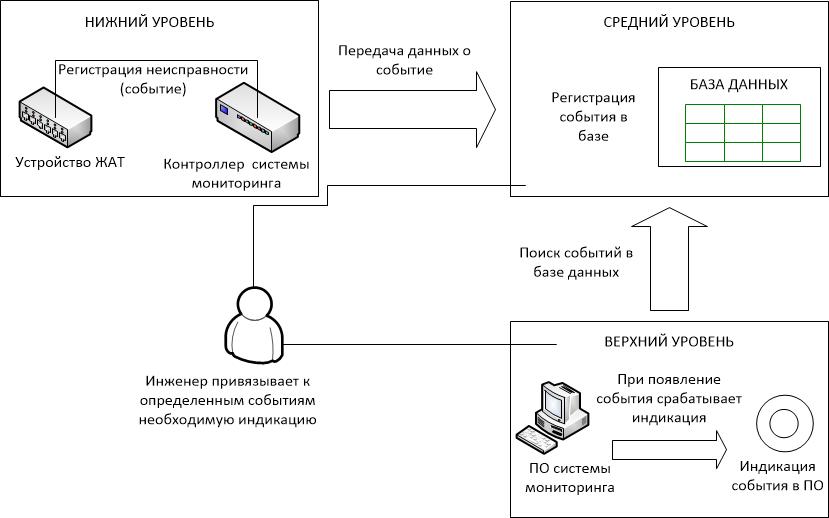


Рисунок 2. Схема привязок.

Например, инженер может привязать к событию падение напряжения индикацию мигание красным. После того как на определенном устройстве упадет напряжение, пользователи ПО верхнему уровню увидят мигание красным на соответствующем устройстве.

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, привязать 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.

**Схема работы программы**

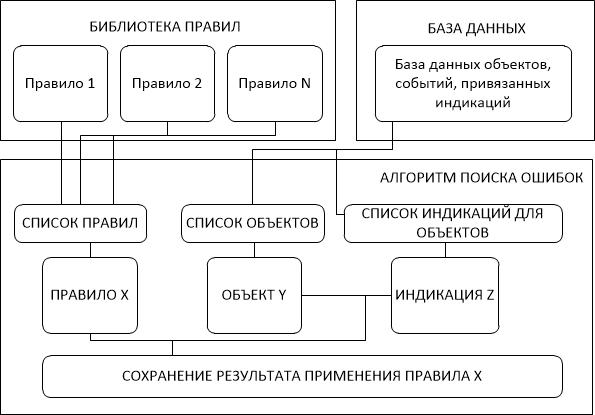


Рисунок 3.Схема работы алгоритма поиска ошибок.

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:**

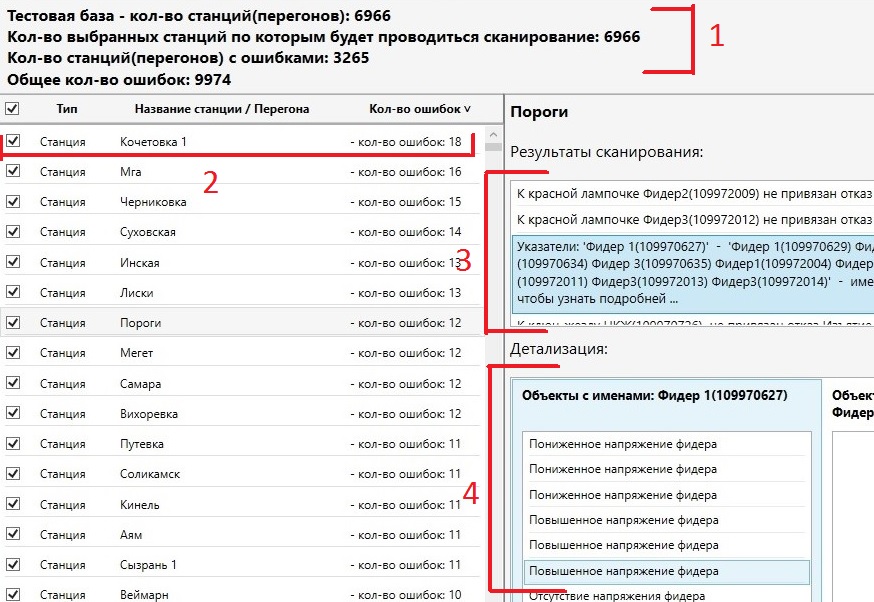


Рисунок 4. Окно вывода результата работы алгоритма.

*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 алгоритмов кластеризации для автоматического построения библиотеки правил.

**Применение алгоритмов кластеризации**

Для дальнейшего усовершенствования программы автоматизации поиска ошибок необходимо:

1. Исследовать алгоритм кластеризации k-means (допустимо ли его использование в конкретном случае).
2. Применить алгоритм k-means для автоматического построения библиотеки плавил.