УТВЕРЖДАЮ

«___»_____ 20 г.

ОПИСАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

ШИФР. "ВЕТРОЛ".ПА.2022

Автоматизированная система

управления турбокомпрессорным агрегатом

Листов 26

СОГЛАСОВАНО
«___»_____20__ г.

РАЗРАБОТАНО
Генеральный директор
ЗАО «ТоксСофт-14»

_____/Т.О. Хазарадзе /
«___»_____20__ г.

2022г

Перечень принятых сокращений

Сокращение	Описание
АРМ	Автоматизированное рабочее место
АСУТП "Ветрол"	Автоматизированная система управления турбокомпрессорным агрегатом
БД	База данных
КТС	Комплекс технических средств
ПО	Программное обеспечение
ТКА	Турбокомпрессорный агрегат

Аннотация

Настоящий документ содержит описания структуры, функций, а также методов и средств разработки программного обеспечения «Программное обеспечение АСУ ТП управления турбокомпрессорными и нагнетательными агрегатами Ю-Скат ВЕТРОЛ (ВЕТРОЛ, VETROL)» (далее *Система* или *"ВЕТРОЛ").*

Оглавление

Введение

В настоящем документе приведено описание программного обеспечения автоматизированной системы управления турбокомпрессорным агрегатом.

Подробное описание технического обеспечения, необходимого для функционирования АСУТП "Ветрол", выходит за рамки работ данного проекта.

Подробное описание алгоритмов приведено в документе «Описание алгоритмов».

Система предназначена для решения следующих задач:

- управления турбокомпрессором
- хранения и использования данных по работе оборудования турбокомпрессора
- автоматизированного построения графиков и отчетов по данным

Система реализована на клиент-серверной технологии.

Клиенты представляют собой Eclipse RCP-приложения реализованные на языке Java. **Eclipse** – это среда разработки, написанная на Java, развиваемая и поддерживаемая Eclipse Foundation (участниками которого являются IBM, SAP, Oracle, компании, предлагающие продукты на основе Eclipse RCP и участвующие в развитии экосистемы Google, RedHat, Adobe, Cisco, Intel). **Eclipse RCP** – это набор плагинов для создания, так называемых, rich client application. Модульность, кроссплатформенность, поддержка мультиязычности, бесплатность, огромное количество существующих плагинов, библиотек и фреймвороков. Все это позволяет создавать приложения коммерческого уровня.

Сервер системы реализован на платформе **Java Platform, Enterprise Edition**, сокращенно **Java EE** — набор спецификаций и соответствующей документации для языка Java, описывающей архитектуру серверной платформы для задач средних и крупных предприятий. JEE является промышленной технологией и в основном используется в высокопроизводительных проектах, в которых необходима надежность, масштабируемость, гибкость.

В качестве реализации JEE в проекте используется **WildFly** (ранее **JBoss Application Server** или **JBoss AS**) — Java EE-сервер приложений с открытым

исходным кодом, разработанный одноимённой компанией. Достаточно хорошая реализация принципов Java EE делает WildFly конкурентом для аналогичных проприетарных программных решений, таких, как WebSphere или WebLogic.

Сервер приложений WildFly — расширяемая, модульная система. Для реализации задач проекта Ветрол-МосКокс в сервер приложений WildFly добавлены модули (java-библиотеки) собственной разработки компании ТоксСофт. **Сервер приложений Ветрол-МосКокс —** это платформа WildFly плюс набор библиотек разработанных ЗАО "ТоксСофт-14".

В качестве СУБД используется **MariaDB** —ответвление от системы управления баз данных MySQL, разрабатываемое сообществом под лицензией GNU GPL. Разработку и поддержку MariaDB осуществляет компания MariaDB Corporation Ab и фонд MariaDB Foundation. **MariaDB** является решением для малых и средних приложений.

Сам программный код Системы реализован на платформе **USkat**. Платформа **uskat** – это средство для разработки систем автоматизации используемое в ЗАО "ТоксСофт-14".

Пояснение: **USkat -**"Платформа для разработки программных систем USkat (Свидетельство о государственной регистрации программы для ЭВМ №2022685071, дата регистрации 21.12.2022)".

В разработке этого проекта из платформы использовалось:

- Программное обеспечение сервера;
- Программное обеспечение клиента;
- Программное обеспечение моста с OPC DA
- Методика проектирования систем автоматизации

1 ПО моста OPCJava.

Физически размещается на персональных компьютерах АРМов пользователей.

Представляет из себя исполняемый java archive реализующий следующий функционал:

- подключение к стороннему OPC серверу по протоколу OPC DA. Для подключения к OPC серверу используется открытая и свободная библиотека JEasyOPC (https://sourceforge.net/projects/jeasyopc/reviews/)
- чтение/запись тегов OPC сервера
- подключение к серверу системы Ветрол
- мапирование тегов OPC на данные и команды системы Ветрол

Программный код моста не требует модификации при изменении набора тегов/данных/команд. Такие вопросы закрываются с помощью изменения конфигурационных файлов моста. Методика конфигурирования описана в Приложении 1 к данному документу.

2 ПО сервера системы.

Физически размещается на персональных компьютерах АРМов пользователей. Все функции сервера сгруппированы в службах, а службы содержатся в одной точке входа под названием ServerAPI. Примеры служб:

- sysdescr — уже упомянутое описание системы, позволяет программно создавать, редактировать и получать описания иерархии классов предметной области;
- objservice — содержит перечень всех объектов системы, по мере работы системы появляются новые объекты, старые исчезают (но остаются в истории!).
- linkservice — управление связями между сущностями модели предметной области;
- userservice — управление пользователями системы и их правами;
- и т. д. Полный список сервисов ПО доступен для просмотра в исходных кодах передаваемых с системой.

3 ПО клиента.

Физически размещается на персональных компьютерах АРМов пользователей системы и связан с сервером.

В системе реализуются следующие типы автоматизированных рабочих мест (АРМ):

- АРМ оператора,
- АРМ разработчика.

Отметим, что рабочее место разработчика (АРМ разработчика) не является частью Системы, и просто представляет собой сконфигурированный набор средств разработки, утилит разработчика и исходные коды Системы.

Реализации любого типа АРМа строится на концепции «контейнер» + «плагины». Контейнер с единым интерфейсом. Реализуются плагины и библиотеки для решения прикладных задач. На данном этапе это следующие задачи:

- библиотечный модуль для связи с сервером системы;
- набор плагинов для реализации функционала АРМов.

Описание особенностей реализации:

- библиотечный модуль не требует доработки при добавлении новых плагинов;
- работает под Linux/Windows 8+;
- средства разработки (интегрированная среда разработки, компиляторы, средства отладки и т.д.) открытые и бесплатные.

4 Методика проектирования систем автоматизации.

Ядро сервера, также как и любой клиентский код системы, оперирует объектной моделью предметной области приложения. Объектная модель описывает все данные и понятия предметной области, с чем работает система, такие как иерархия типов (классов — термин используемый в технологии «Объектно-ориентированное проектирование») объектов. В частности, все существующие объекты автоматизации представлены как типизированные объекты. Например, типами (классами) являются "реверсивный двигатель", "модуль аналогово ввода", "пользователь" и т. п. Полное описание классов зависит от конкретного проекта будет приведено в формате файла электронной таблицы при сдаче системы.

Каждый тип (класс) описывается как совокупность следующих свойств:

- ***атрибут*** — неизменяемые во времени параметры объекта. Например, тип (класс) "пользователь" имеет атрибут "Фамилия". У разных объектов этого типа отличаются значения атрибута: "Иванов", "Петров и т.п.;
- ***связь*** — взаимосвязь, который объект данного типа имеет с другими объектами разных типов. Например, объект типа "турбокомпрессорный агрегат" имеет связь типа "содержит в себе" с объектами типов "воздушный контур" , "масляный контур" и др.;
- ***данное*** — это свойства, значение которых меняется во времени. Например, "аналоговый вход" имеет данное "текущее значение" и т. п. Здесь необходимо отметить что АСУТП "Ветрол" является системой реального времени и коды программы оптимизированы под работу с такими данными с точки зрения объектно-ориентированного проектирования;
- ***событие*** — описывает то, что происходит с объектами. Можно сказать, что объекты предметной области "генерируют" события. Например, при подключении нового пользователя система генерирует событие "Пользователь Иванов подключился в 10:00 02/02/19";
- ***команда*** — описывает то, что может делать указанный объект по команде Системы. Все управляющие команды которые поддерживает данный тип

описываются в системе.

Описание системы и хранение свойств объектов реализуют сервис классов и сервис объектов сервера Системы.

5 Структура программного обеспечения

Структура ПО приведена на рисунке 1.

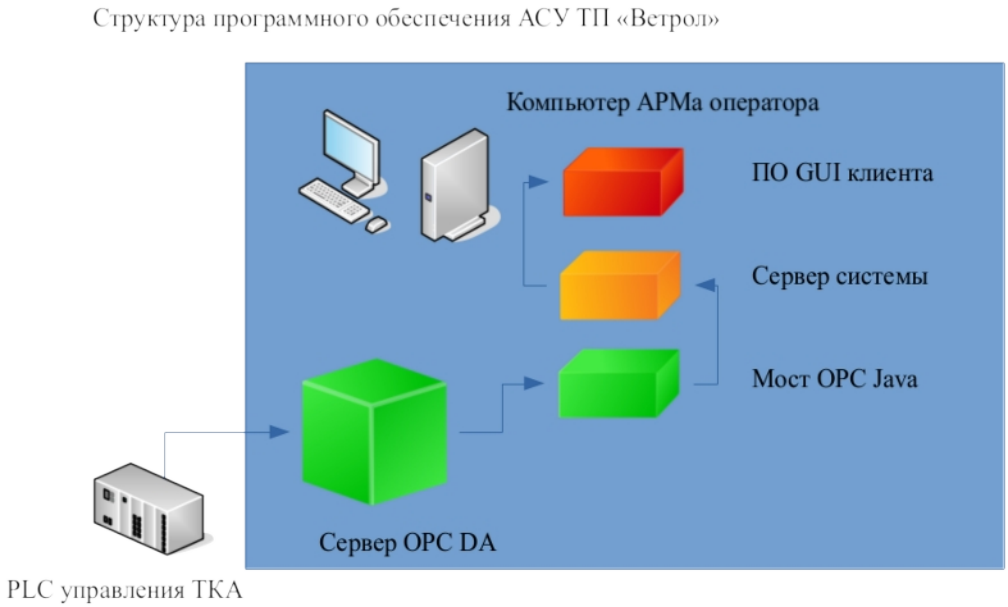


Рисунок 1: Структура ПО системы

Программное обеспечение всех частей Системы полностью написано на одном языке программирования — Java и физически работает на компьютере АРМа оператора.

На рисунке обозначены:

- зеленый куб — программное обеспечение Siemens;
- зеленый параллелепипед — мост OPC2Java;
- желтый параллелепипед — сервер системы "Ветрол";
- красный параллелепипед — программное обеспечение АРМа оператора

ПО системы состоит из следующих частей:

- **Сервер системы "Ветрол"** — центральная часть системы, базовый программный компонент инфраструктуры. Все данные реального времени

хранятся не менее 3 лет в оперативном доступе (без переноса в архив). Сервер системы "Ветрол" устроен так, что глубина хранения данных в оперативном доступе не влияет на производительность работы Системы;

- **ПО GUI** — программа ЧМИ;
- **ПО моста OPC Java** - мост между OPC DA сервером и системой Ветрол.

6 Функции частей программного обеспечения

6.1 Сервер "Ветрол"

6.1.1 Функции

Центральный компонент системы, работает в режиме 24x7 и осуществляет хранение всех данных (оперативное и долговременное).

6.2 АРМ

Средство взаимодействия (человека-машинного интерфейса) с пользователем, осуществляет визуализацию: просмотр данных по работе ТКА с помощью мыши, клавиатуры и опционально других средств взаимодействия (сенсорный экран, планшет).

6.3 Шлюз с сервером OPC DA (Мост)

Программная компонента взаимодействия с внешней системой по протоколу OPC DA. С одной стороны («*правая сторона*») взаимодействет с платформой S5 (понятия ОО описания системы), а с другой стороны («*левая сторона*») — с OPC DA . Служит «переводчиком», мостом между системами

6.3.1 Мост сопряжения с OPC DA

В составе поставки системы входят стандартный мост

Взаимодействие с OPC-сервером — полностью конфигурируемый мост обмена данными в виде тегов в реальном времени с любым OPC-сервером по протоколу OPC DA.

По запросу заказчика дополнительно возможно создание других мостов (например, как мост публикации API).

6.4 Подсистемы:

Программное обеспечение АСУТП "Ветрол" включает с себя следующие подсистемы:

- подсистема описания модели предметной области;
- работа с текущими данными;
- работа с хранимыми данными;

- работа с командами;
- работа с мнемосхемами;
- подсистема создания отчетов/графиков;
- подсистема создания и обработки запросов на выборку данных.

7 Методы и средства разработки программного обеспечения подсистем

Система разрабатывается с помощью технологии объектно-ориентированного программирования с использованием языка программирования Java SE 17. В качестве СУБД используется MariaDB (диалект MySQL) версии 10.6. При создании интерактивного пользовательского интерфейса применяются фреймворк Eclipse RCP.

При разработке программных компонент использовались следующие средства:

1. интегрированная среда разработки Eclipse 2021-12 (4.22.0);
2. система контроля версий Git;
3. средство разработки JDK SE Development Kit 17.

8 Операционная система

- Для работы с системой на клиентских местах используется Windows 8+ или Linux.

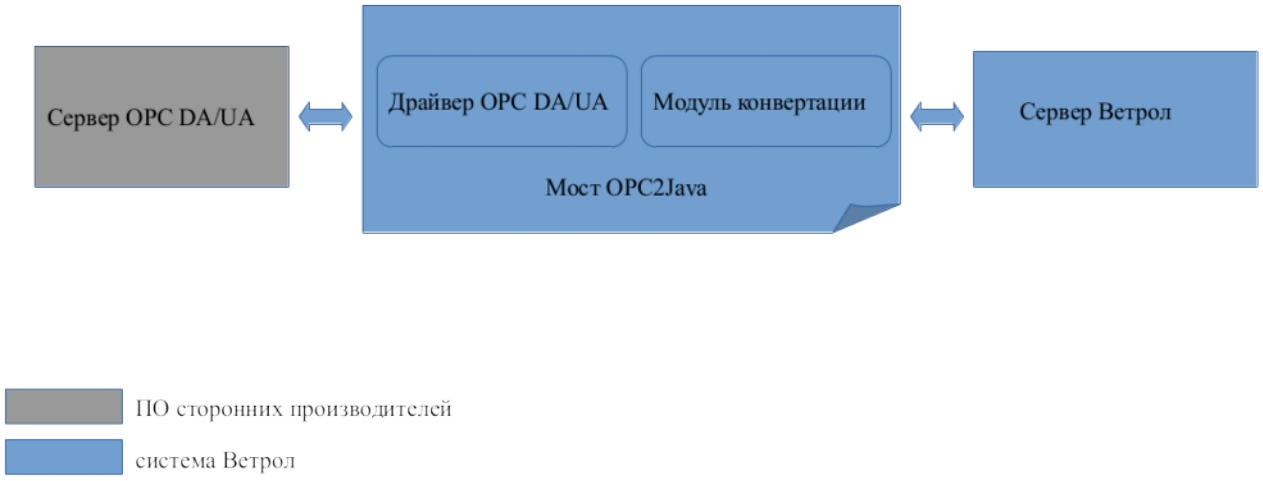
9 Средства, расширяющие возможности операционной системы

При разработке АСУТП «Ветрол»не было использовано средств, расширяющих возможности операционной системы.

Приложение 1. Руководство по настройке моста OPC2Java

В настоящем приложении приведено описание настройки специальной программы «Мост OPC2Java", предназначенной для обмена данными между OPC-сервером и сервером приложений системы.

1 Общая информация



Мост OPC2Java в системе "ВЕТРОЛ" является отдельной программой, написанной на языке Java и предназначенной для обмена данными между OPC-сервером и сервером "ВЕТРОЛ". Мост OPC2Java выполняет роль клиента как OPC-сервера, так и сервера "ВЕТРОЛ". Мост OPC2Java построен на основе программы «нижнего уровня» (НУ). Программа НУ предназначена для сбора данных с различных устройств с целью первичной обработки и передачи их на сервер и для передачи управляющих сигналов с сервера на устройства. В этих терминах OPC-сервер является устройством. Одной из главных задач программы НУ является

сопоставление (мапирование, связывание) конкретных данных устройств и данных сервера, а также указание способа их конвертации, первичной обработки (в обе стороны). Программа НУ связывает данные устройств и данные сервера с помощью конфигурационных файлов.

2 Архитектура моста

Архитектурно программа НУ состоит из нескольких частей, основными из которых являются две. Первая часть — набор «драйверов», непосредственно работающих с устройствами. Вторая часть — набор модулей, функцией которых является конвертация данных устройств в сущности сервера, а также связывание (мапирование) данных сервера и данных устройств. Все части НУ (включая упомянутые две основные) управляются одним общим циклом.

Мост OPC2Java системы "ВЕТРОЛ" с точки зрения построения программы НУ имеет один «драйвер», работающий как клиент OPC, и один модуль мапирования и конвертации данных OPC <-> сервер "ВЕТРОЛ", работающий ка клиент сервера "ВЕТРОЛ". Каждому из них соответствует свой конфигурационный файл.

3 Описание данных

3.1 Описание данных OPC -сервера

Единицей описания данных OPC-сервера является тег. С точки зрения конфигурирования «драйвера» моста все теги делятся на теги для чтения и теги для записи. Один и тот же тег может являться как тегом для чтения, так и тегом для записи. Теги на чтение делятся на синхронные и асинхронные. Чтение синхронных тегов производится периодически через равные интервалы времени. Чтение асинхронного тега производится при поступлении сигнала с OPC-сервера об изменении значения этого тега.

С точки зрения конфигурации теги могут быть следующих типов:

Integer;

Boolean;

Float;

String.

Типы тегов при конфигурации должны соответствовать типам соответствующих тегов на OPC-сервере.

3.2 Описание данных сервера "ВЕТРОЛ"

Структура данных сервера "ВЕТРОЛ" разработана в соответствии с принципами объектно-ориентированного подхода.

Все сущности предметной области сгруппированы по классам. Класс содержит описание соответствующих ему сущностей. В контексте работы моста OPC2Java главными характеристиками сущностей, описанными в классах, являются данные, команды и события.

Описание данного состоит из строкового идентификатора, признака, что данное является текущим, признака, что данное является историческим, типом значения данного, признаком, что данное является синхронным (в противном случае – асинхронным). Полным идентификатором данного в системе является совокупность трёх строковых идентификаторов: идентификатора класса сущности, идентификатора экземпляра сущности, идентификатора данного (в классе).

Описание команды состоит из строкового идентификатора, строковых идентификаторов и типов значений параметров команды. Полным идентификатором команды в системе является совокупность трёх строковых идентификаторов: идентификатора класса сущности, идентификатора экземпляра сущности, идентификатора команды (в классе).

Описание события состоит из строкового идентификатора, строковых идентификаторов и типов значений параметров события. Полным идентификатором события в системе является совокупность трёх строковых идентификаторов: идентификатора класса сущности, идентификатора экземпляра сущности, идентификатора события (в классе).

Задачей моста является перевод значений тегов на чтение в данные и события сущностей системы и перевод команд в значения тегов на запись.

4 Принцип конфигурирования моста

Задачей конфигурирования является сопоставление тегов на чтение — данным и событиям сущностей, тегов на запись — командам сущностей.

Конфигурационный файл драйвера («*.devcfg») содержит список тегов, разделённый на три раздела: теги с синхронным чтением, теги с асинхронным чтением, теги на запись. Описание каждого тега содержит его идентификатор на OPC-сервере, тип значений тега и идентификатор, который будет использоваться внутри моста (НУ). Конфигурационный файл драйвера определяет то, какие данные мосту следует считывать с OPC-сервера и передавать на OPC-сервер, и в каком формате.

Конфигурационный файл модуля конвертации («*.dlmcfg») состоит из четырёх разделов: данных, описания команд, команд, событий.

Описание одного данного сервера содержит: идентификатор класса сущности, идентификатор экземпляра сущности, идентификатор данного, идентификатор тега на чтение, указание обработчика значения тега, дополнительные параметры данного сервера (историческое, синхронное и т.д.).

Описание одной команды сервера содержит: идентификатор класса сущности, идентификатор экземпляра сущности, идентификатор команды, идентификатор тега на запись, указание обработчика команды, формирующего значение тега, идентификаторы параметров команды сервера.

Описание одного события сервера содержит: идентификатор класса сущности, идентификатор экземпляра сущности, идентификатор события, идентификатор тега на чтение, указание формирователя события, указание определителя события, указание формирователя значений параметров события, идентификаторы параметров события сервера.

Конфигурационный файл модуля конвертации определяет мапирование (связывание, сопоставление) конкретных тегов на конкретные данные (команды, события) сервера и способ их конвертации.

5 Примеры конфигурирования моста

В примерах конфигурации моста, представленных далее, присутствуют указания на конкретные java-классы. Эти java-классы покрывают все потребности текущей конфигурации системы "ВЕТРОЛ". При выявлении недостаточности этих реализаций или при появлении новых требований к системе библиотека

соответствующих java-классов может быть расширена. Тогда при конфигурации нужно будет указывать другие, соответствующие новым требованиям, java-классы.

5.1 Добавление тега

Чтобы добавить тег в конфигурацию «драйвера» моста OPC2Java следует вставить следующую структуру в один из разделов конфигурационного файла «mcc.devcfg» (в зависимости от того какой тег добавляется: синхронный тег на чтение, асинхронный тег на чтение, тег на запись):

```
{
   false,
   "pin.tag.SIMATIC_300_1__CPU_314C_2_PtP_P62_CV.def",
   {
     opc.tag="SIMATIC 300(1).CPU 314C-2 PtP.P62.CV",
     pin.id="",
     pin.type="Float"
   },
   {
   }
},
```

здесь:

верхняя строка — уникальное название структуры — обязательная строка, необходимая только в рамках формата файла;

opc.tag — идентификатор тега на OPC-сервере, он же используется как идентификатор тега внутри моста OPC2Java;

pin.id — неиспользуемое поле;

pin.type – тип значения тега (то, как его будет интерпретировать мост OPC2Java);

самое верхнее булевое поле должно иметь значение false, означающее, что

структура не является массивом, - обязательное поле в рамках формата файла.

5.2 Добавление данного сервера "ВЕТРОЛ"

Для добавления одного данного в конфигурацию модуля моста OPC2Java следует вставить в раздел данных конфигурационного файла «mcc.dlmcfg» структуру следующего вида:

```
{
   false,
   "data.SIMATIC_300_1__CPU_314C_2_PtP_P62_CV.def",
   {
    pin.id="",
    java.class=
"ru.toxsoft.l2.dlm.opc_bridge.submodules.data.OneToOneDataTransmitterFactory",
    class.id="mcc.AnalogInput",
    obj.name="n2AI_P62",
    data.id="rtdCurrentValue",
    tag.dev.id="opc2s5.bridge.collection.id",
    tag.id="SIMATIC 300(1).CPU 314C-2 PtP.P62.CV",
    is.hist=true,
    is.curr=true,
    synch.period=1000
   },
   {
   }
},
```

здесь:

верхняя строка — уникальное название структуры — обязательная строка,

необходимая только в рамках формата файла;

pin.id — неиспользуемое поле;

java.class — java-класс, занимающийся конвертацией и трансляцией значения тега в значение данного сервера;

class.id, obj.name, data.id — совокупность трёх строковых идентификаторов: - идентификатор класса сущности, идентификатор экземпляра сущности, идентификатор данного сущности - которые однозначно указывают на одно данное сервера;

tag.dev.id — идентификатор устройства (OPC-сервер), откуда поступает значение данного (этот идентификатор указан в корневом разделе файла «mcc.devcfg»: id="opc2s5.bridge.collection.id");

tag.id - идентификатор тега внутри моста OPC2Java;

is.hist – признак того, что данное является историческим;

is.curr — признак того, что данное является текущим;

synch.period — период синхронизации в миллисекундах;

самое верхнее булевое поле должно иметь значение false, означающее, что структура не является массивом, - обязательное поле в рамках формата файла.

В случае если значение добавляемого данного формируется из одного бита Integer значения тега («слово состояния» - набор битов состояния), в описание данного следует добавить поле bit.index — номер читаемого бита.

5.3 Добавление команды

Для добавления одной команды в конфигурацию модуля моста OPC2Java следует вставить в раздел команд («cmdDefs») конфигурационного файла «mcc.dlmcfg» структуру следующего вида:

```
{
  false,
  "cmd.cmdX0.def",
  {
```

```
        class.id="mcc.AnalogInput",
        obj.name="n2AI_P62",
        cmd.id="cmdX0",
        command.exec.java.class=
    "ru.toxsoft.l2.dlm.opc_bridge.submodules.commands.ValueCommandExec",
        value.param.id="value",
        tag.dev.id="opc2s5.bridge.collection.id",
        tag.id="SIMATIC 300(1).CPU 314C-2 PtP.P62.X0"
      },
      {
      }
    },
```

здесь:

верхняя строка — уникальное название структуры — обязательная строка, необходимая только в рамках формата файла;

class.id, obj.name, cmd.id — совокупность трёх строковых идентификаторов: - идентификатор класса сущности, идентификатор экземпляра сущности, идентификатор команды сущности - которые однозначно указывают на одну команду сервера;

command.exec.java.class — java-класс, занимающийся обработкой команды и трансляцией значения в тег OPC-сервера;

value.param.id — идентификатор параметра команды;

tag.dev.id — идентификатор устройства (OPC-сервер), откуда поступает значение данного (этот идентификатор указан в корневом разделе файла «mcc.devcfg»: id="opc2s5.bridge.collection.id");

tag.id - идентификатор тега на запись внутри моста OPC2Java;

самое верхнее булевое поле должно иметь значение false, означающее, что

структура не является массивом, - обязательное поле в рамках формата файла.

В случае если добавляемая команда должна изменять не всё значение тега, а один бит Integer значения тега («контрольное слово» - набор управляющих битов), в описание команды следует добавить поле bit.index — номер изменяемого бита.

При добавлении команды следует убедиться (и при необходимости добавить), что в разделе «cmdClassDefs» файла «mcc.dlmcfg» в описании класса присутствуют идентификатор соответствующей команды и идентификатор соответствующего экземпляра сущности:

```
{
  false,
  "class.mcc.AnalogInput.def",
  {
   class.id="mcc.AnalogInput",
   obj.names.list="n2AI_P62,n2AI_P61,n2AI_TM2,n2AI_TM1,...",
   cmd.ids.list="cmdFilterConst,cmdX0,cmdY0,cmdX1,..."
  },
  {
   }
}
```

здесь:

верхняя строка — уникальное название структуры — обязательная строка, необходимая только в рамках формата файла;

class.id – идентификатор класса сущности, чья команда добавляется в конфигурацию модуля моста;

obj.names.list – список идентификаторов экземпляров указанной сущности;

cmd.ids.list – список идентификаторов команд указанной сущности.

5.4 Добавление события

Для добавления одного события в конфигурацию модуля моста OPC2Java следует вставить в раздел событий («eventDefs») конфигурационного файла «mcc.dlmcfg» структуру следующего вида:

```
{
  false,
  "event.evtSetPoint3indication.def",
  {
    class.id="mcc.AnalogInput",
    obj.name="n2AI_T3",
    event.id="evtSetPoint3indication",
    event.sender.java.class=
"ru.toxsoft.l2.dlm.opc_bridge.submodules.events.OpcTagsEventSender",
    condition.java.class=
"ru.toxsoft.l2.dlm.opc_bridge.submodules.events.OneTagSwitchEventCondition",
    condition.switch.on=true,
    condition.switch.off=true,
    bit.index=3,
    tag.dev.id="opc2s5.bridge.collection.id",
    tag.id="SIMATIC 300(1).CPU 314C-2 PtP.T3.CW",
    param.former.java.class=
"ru.toxsoft.l2.dlm.opc_bridge.submodules.events.OneTagToOneParamFormer",
    former.event.params="on"
  },
   {
    }
```

},

здесь:

верхняя строка — уникальное название структуры — обязательная строка, необходимая только в рамках формата файла;

class.id, obj.name, event.id — совокупность трёх строковых идентификаторов:- идентификатор класса сущности, идентификатор экземпляра сущности, идентификатор события сущности - которые однозначно указывают на одно событие сервера;

event.sender.java.class — java-класс, формирующий событие и отправляющий его на сервер;

condition.java.class — java-класс, анализирующий изменение значения тега и принимающий решение о формировании события;

condition.switch.on — признак того, что событие формируется по прямому фронту сигнала тега;

condition.switch.off — признак того, что событие формируется по обратному фронту сигнала тега;

bit.index — номер анализируемого бита, если тип значения тега — Integer (поле не нужно, если тип значения тега — Boolean);

tag.dev.id — идентификатор устройства (OPC-сервер), откуда поступает значение данного (этот идентификатор указан в корневом разделе файла «mcc.devcfg»: id="opc2s5.bridge.collection.id");

tag.id – идентификатор анализируемого тега на чтение внутри моста OPC2Java;

param.former.java.class - java-класс, формирующий значения параметров события;

former.event.params — идентификаторы параметров события;

самое верхнее булевое поле должно иметь значение false, означающее, что структура не является массивом, - обязательное поле в рамках формата файла.