МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет»
(национальный исследовательский университет)
Высшая школа экономики и управления
Кафедра «Информационные технологии в экономике»

РАБОТА ПРОВЕРЕНА
Рецензент, ведущий программист
_____/ И.А. Кожевников /
«_____» _____2019 г.

ДОПУСТИТЬ К ЗАЩИТЕ
Зав. кафедрой, д.т.н., с.н.с.
_____/ Б.М. Суховилов /
«_____»_____2019 г.

РАЗРАБОТКА СЕРВЕРА КОММУНИКАЦИЙ ДЛЯ СИСТЕМЫ МОНИТОРИНГА СОСТОЯНИЯ ОБОРУДОВАНИЯ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.04.03.2019.301/65. ВКР

Руководитель, к.т.н., доцент
_____ / В. А. Конов /
«_____» _____2019 г.

Автор
студент группы ЭУ-235
_____ / Е.Л. Горвиц /
«_____» _____2019 г.

Нормоконтролер, ст. преподаватель
_____ / Е. Н. Горных /
«_____»_____2019 г.

Челябинск, 2019

АННОТАЦИЯ

Горвиц Е.Л.

Разработка системы мониторинга состояния оборудования – Челябинск: ЮУрГУ, ЭУ-235, 57 стр., 27 рис., список литературы – 29 наим.

Выпускная квалификационная работа посвящена проектированию, реализации и тестированию OPC Unified Architecture сервера для Genius CM, интеллектуальной системы мониторинга состояния оборудования.

В работе проведен анализ предметной области и обзор аналогов, рассмотрены системы мониторинга и сбора технологических данных, протоколы промышленной коммуникации. Приведено сравнение существующих аналогов и разрабатываемого решения. Выполнен анализ требований к системе, выявлены основные классы, построена доменная модель. В тексте работы подробно рассмотрен процесс реализации сервисов приложения в соответствии с спецификациями OPC и их тестирование.

Интеграция приложения с системой Genius CM позволяет предоставить унифицированный доступ к уже имеющимся и поступающим в онлайн режиме от систем мониторинга данным. Это позволяет использовать систему Genius CM широкому кругу клиентских приложений, встроить её в единую информационную сеть предприятия.

ОГЛАВЛЕНИЕ

ГЛОССАРИЙ

Alarms & Events service (AE) – OPC-сервис, предоставляющий информацию о тревогах и нештатных ситуациях.

Data Access service (DA) – OPC-сервис, предоставляющий доступ к текущим (онлайн) показателям оборудования.

Genius Condition Monitoring System (Genius CM) – программное обеспечение для мониторинга состояния оборудования литейных машин.

Historical Data Access service (HDA) – OPC-сервис, предоставляющий доступ к истории наблюдаемых значений.

Human-Machine Interface (HMI) – Человеко-машинный интерфейс. Обеспечивает взаимодействие человека-оператора с управляемыми им машинами.

Machine-to-Machine communication (M2M) – Межмашинный интерфейс. Обеспечивает взаимодействие по принципу «Машина-Машина» или «ПО-Машина».

OPC Unified Architecture (OPC UA) – новая версия OPC стандарта, созданная в качестве замены устаревшему OPC Classic.

Open Platform Communications (OPC) – M2M интерфейс для управления объектами автоматизации и технологическими процессами.

Process Data Acquisition System (PDA) – система сбора технологических данных.

Programming Logical Controller (PLC) – программируемый логический контроллер.

ВВЕДЕНИЕ

Актуальность темы

Быстрое развитие современных технологий требует непрерывной модернизации производства. Компании, владеющие заводами и фабриками, заинтересованы в переходе на производственный стандарт Industry 4.0. Стандарт подразумевает дигитализацию предприятия: широкое внедрение цифровых технологий в производственные процессы, организацию «умного» мониторинга, постоянный анализ технологических данных. В основе этих процессов лежит взаимодействие цифровых устройств – Machine-to-Machine Communication (M2M).

Новое программное обеспечение должно предоставлять не только HMI, но и M2M интерфейс. Приложение-сервер, следующее спецификации OPC, является одним из возможных способов организации такого интерфейса. Необходимость разработки подобного приложения обуславливает актуальность темы.

Краткое содержание работы

В работе рассматривается проектирование и реализация OPC Unified Architecture сервера для Genius CM, системы мониторинга состояний компании SMS Group. Разработка ведется на языке программирования C# 6.0 с использованием UA .NET StandardLibrary Stack – официального фреймворка OPC Foundation. Интеграция приложения с системой Genius CM позволит предоставить унифицированный доступ к уже имеющимся и поступающим в онлайн режиме от систем мониторинга данным. Это позволит использовать систему более широкому кругу клиентских приложений.

Цели и задачи

Целью работы является разработка OPC UA сервера для системы Genius CM. Для достижения поставленной цели необходимо решить следующие задачи:

1. Проектирование приложения. Необходимо описать возможные варианты использования, выявить функциональные и нефункциональные требования к программному обеспечению. Разработанная система должна иметь легко поддерживаемую архитектуру, которая может быть расширена в дальнейшем.

2. Реализация OPC-сервисов, в соответствии со спецификациями: Data Access (DA) сервис для получения данных от станций анализа, Historical Data Access (HDA) сервис, обеспечивающий доступ к истории наблюдаемых значений, и Alarms & Events (AE) сервис для информирования о нештатных ситуациях.

3. Реализация методов управления станциями наблюдения: включение и отключение станции, запрос её состояния.

Структура и объем работы

Выпускная квалификационная работа состоит из введения, четырех основных разделов, заключения, списка литературы и 3 приложений. Объем работы составляет 57 страниц, объем библиографии – 29 наименований.

Первый раздел посвящен анализу предметной области. Приводится обзор современных методов получения технологических данных из разнообразных источников. Рассматриваются существующие аналоги и система Genius CM. Также приведено описание стандарта OPC и его новой версии – спецификации OPC UA.

Второй раздел описывает проектирование архитектуры OPC UA сервера. Формулируются требования к системе, выявляются варианты использования. В разделе приведена доменная диаграмма приложения, описывающая основные сущности нижележащей системы, модели и основные объекты OPC.

Третий раздел посвящен реализации приложения. Подробно рассматривается спецификация и реализация каждого сервиса. Приводятся фрагменты исходного кода и конфигурации приложения. Также в разделе описывается взаимодействие сервера с Genius CM WebAPI.

Четвертый раздел содержит описание процесса тестирования приложения. Рассматриваются использованные инструменты, приводится протокол тестирования.

В заключении приводится анализ результатов работы и рассматриваются возможные пути дальнейшего развития системы.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Современная металлургия и производство стали являются сложными техническими процессами. Они предъявляют высокие требования к используемым системам контроля и мониторинга. Процесс литья зависит от большого количества параметров на разных стадиях: химический состав материала, параметры производства и переменные состояния, температура жидкой стали, давление воздуха, скорость литья, параметры охлаждения и многие другие [20].

Даже небольшое отклонение значения одного из параметров от допустимого может оказать серьезное влияние на качество продукции, а в исключительных случаях стать причиной повреждения оборудования или создать опасность для персонала. Доступ данным мониторинга необходим для быстрого принятия решений и является критически важным для металлургических предприятий.

В дополнение к онлайн-данным также важен доступ к истории значений каждого параметра. Исторические данные необходимы для методов машинного обучения и прикладной статистически, которые широко используются для долгосрочного анализа и оптимизации производственных процессов. При этом возникает несколько проблем:

- Существует большое количество разных датчиков. Они могут быть аналоговыми или цифровыми, передавать данные по медному проводу, оптоволокну, WLAN или Bluetooth.
- Характер и объем данных также различен: от нескольких значений в час до нескольких десятков тысяч значений в секунду.
- Клиенты также представлены широким списком различных устройств: встраиваемые контроллеры, станции мониторинга, персональные компьютеры и ноутбуки, мобильные устройства;
- Завод может иметь сложную многоуровневую структуру с большим количеством взаимосвязанных устройств.

Возникает необходимость в комплексном решении, которое может обрабатывать данные от различных сенсоров и предоставлять результаты в универсальном, удобном для пользователя формате. Примером такого решения является IBA Monitoring System [5].

1.1 IBA Monitoring System

Система IBA представляет собой комплекс программного и аппаратного обеспечения, предназначенный для сбора, записи и обработки технологических данных. Система широко поддерживается производителями заводских установок и высокотехнологичного оборудования, что значительно упрощает ее внедрение на производство. Благодаря модульной архитектуре и простой конфигурации, IBA легко масштабируется и может быть адаптирована под различные задачи. Общая структура системы представлена на рисунке 1.

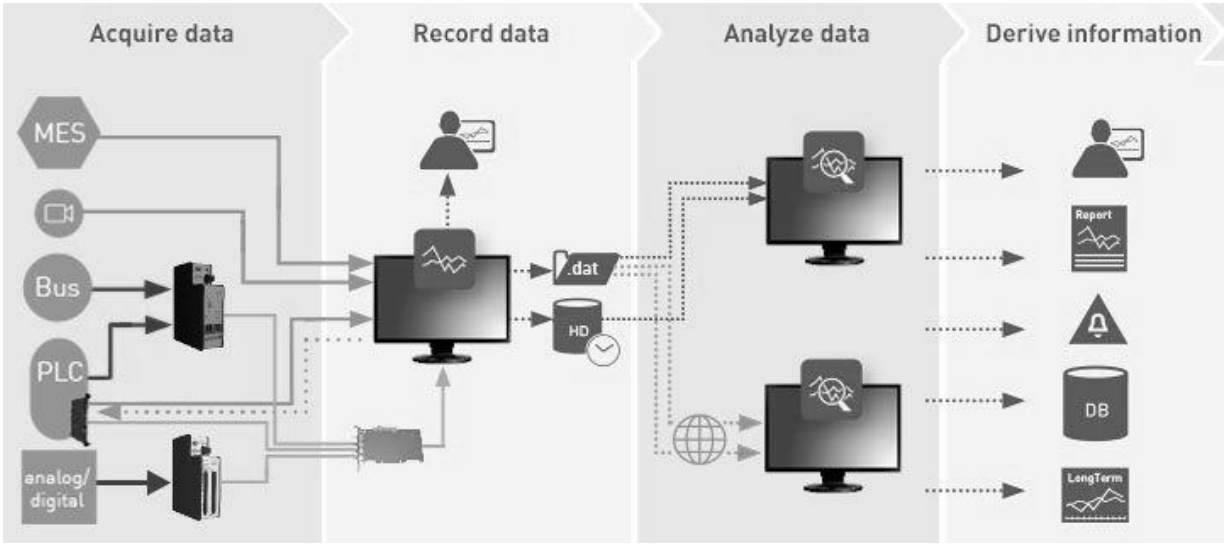


Рисунок 1 – Общая структура IBA Monitoring System

Многие устройства поддерживают технологию IBA-Request. Она позволяет получать значения внутренних переменных контроллера без необходимости его остановки. Доступ к значению осуществляется по его имени, логика изменения содержится в драйвере, поставляемым производителем либо самой IBA. После установки драйвера, данные с устройства будут поступать в IBA Monitoring System. IBA поддерживает долгосрочное хранение данных во внутренней БД. Данные могут записываться непрерывно, по таймеру либо при наступлении на-

страиваемого события. Система способна обрабатывать данные поступающие от сенсоров, шин данных, PLC, MES и других аналоговых и цифровых устройств на высоких скоростях: аналоговые данные – до 100 кГц [7], цифровые – до 1 млн. значений в секунду.

Основным компонентом является станция IBA PDA [6]. Станция является упрощенным персональным компьютером под управлением Unix-подобной операционной системы и отвечает за сбор и сохранение данных от подключенных к ней устройств. IBA PDA может генерировать экстренные извещения при определенных условиях. Например, при существенном отклонении отслеживаемого параметра от нормы. Также станция позволяет проводить диагностику оборудования и выполнять простую визуализацию данных.

Производственные линии оборудованы обычно комплектуются несколькими IBA PDA, для получения данных с разных этапов производства. На рисунке 2 показаны позиции 5 станций металлопрокатного стана: одна на печи, три на промежуточных станках и одна на чистовом прокате.

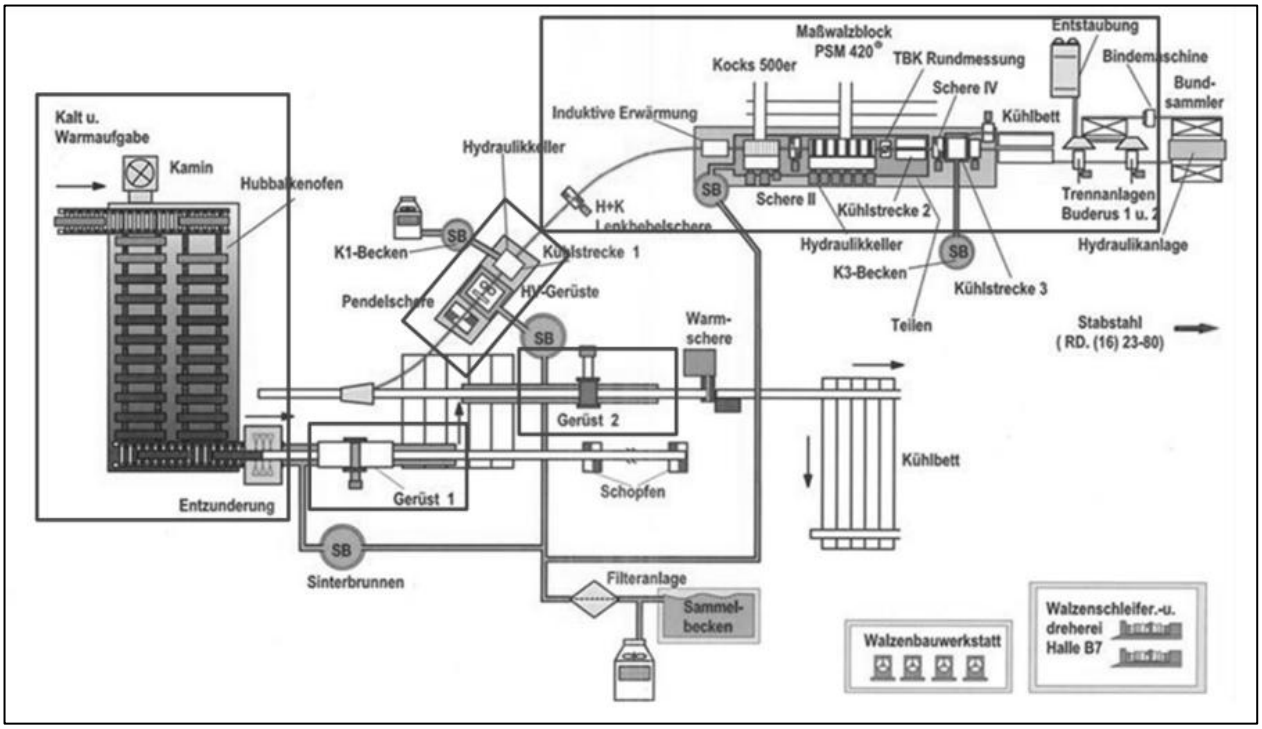


Рисунок 2 – Схема расположения IBA PDA на производственной линии

Данные, полученные от станций, сохраняются в формате .iba. Один файл содержит около 60 минут записи, разделенной на кадры. Новый кадр добавляется

к записи 4 раза в секунду и хранит данные о процессе литья на текущий момент. Данные могут быть получены в режиме реального времени от датчиков (например, текущая температура изделия, уровень шума и вибрации), заданы оператором станка (скорость разливки, размер получаемой заготовки) или рассчитаны с помощью стороннего ПО (ожидаемое время, количество затраченных ресурсов).

Однако IBA Monitoring System имеет несколько недостатков. Система является проприетарной, ее исходный код не доступен сторонним разработчикам. Кроме того, аналитические компоненты IBA дороги в обслуживании [4], поэтому многим компаниям выгоднее использовать ее только для извлечения данных, инвестируя в разработку собственного программного обеспечения для дальнейшей обработки.

В дополнение к низкоуровневой системе получения данных непосредственно от контроллеров и оборудования, необходим протокол более высокого уровня для обмена информацией между устройствами и управления производством. Именно с этой целью был разработан стандарт OPC.

1.2 OPC Classis Standard

OPC является стандартом безопасного взаимодействия и обмена данными между объектами промышленной автоматизации [8]. Стандарт независим от платформы, поэтому устройства разных производителей могут обмениваться информацией. Спецификация OPC была основана на технологиях OLE, COM и DCOM, разработанных Microsoft для семейства операционных систем Microsoft Windows [10]. Она определяет унифицированный набор объектов, интерфейсов и методов, которые могут быть использованы сторонними приложениями, и облегчает их взаимодействие. Объекты остаются неизменными независимо от типа и источника данных.

Разработка и обновление спецификаций OPC координируется международной некоммерческой организацией OPC Foundation [27], созданной в 1994 году ведущими производителями средств промышленной автоматизации.

Протокол OPC был разработан для построения «моста» между приложениями на базе ОС Windows и аппаратными системами управления производством. OPC-сервер аппаратного устройства предоставляет для OPC-клиента унифицированные методы доступа к своим данным, одинаковые для всех устройств и клиентов. Цель состояла в том, чтобы уменьшить количество дублирующих усилий, требуемых от производителей оборудования, SCADA и HMI систем, и помочь им взаимодействовать друг с другом.

Как только производитель оборудования разрабатывает OPC-сервер для нового аппаратного устройства, любой конечный пользователь получает возможность взаимодействовать с ним в своих приложениях. Производитель SCADA, наоборот, может разработать свой OPC-клиент, осуществляющий управление новым или ещё не созданным оборудованием, основываясь на его OPC сервере [24].

Основанный на COM/DCOM, OPC хорошо подходит для работы в распределенных системах, однако эти же технологии легли в основу основных недостатков стандарта:

- OPC-приложение способно работать только на компьютере под управлением ОС Windows. Технологии OLE, COM/DCOM не поддерживаются ни одной другой операционной системой;
- Низкая защищенность приложений, полное отсутствие уровней пользовательских прав и ролей. Если пользователь получил доступ к серверу, он может совершать любые операции с данными и оборудованием, вне зависимости от своего уровня допуска;
- COM и DCOM содержат большое число известных «багов», но работы по их устранению ведутся крайне медленно или не ведутся вовсе. Технологии являются проприетарными, все права принадлежат Microsoft. Программисты не имеют возможности устранять ошибки самостоятельно и вынуждены учитывать их при разработке ПО.

1.3 OPC Unified Architecture

OPC UA является протоколом M2M взаимодействия, разработанным OPC Foundation в качестве замены OPC Standard [22]. Первая версия протокола была представлена в 2006 г. OPC UA имеет все преимущества предыдущей версии и, кроме того, открывает новые возможности использования OPC. Основными нововведениями стали:

- Полностью свободная реализация (лицензия GPL v2 [3]);
- Сервисно-ориентированный подход, уход от монолитной архитектуры;
- Нацеленность на взаимодействие промышленного оборудования и система сбора/контроля данных;
- Улучшение безопасности, введение пользовательских прав;
- Поддержка сложных объектов и структур при разработке моделей.

OPC UA предоставляет набор независимых спецификаций. Каждая описывает набор объектов и методов, необходимых для определенной сферы работы приложения. При этом OPC UA приложение не обязано поддерживать их все [11]. Наиболее часто востребованы три спецификации:

1) Data Access (DA) описывает процесс обмена данными между PLC и другими аппаратными устройствами в режиме реального времени. Эта спецификация зачастую является наиболее важной, так как она позволяет получать онлайн-данные с сенсоров и осуществлять управление ими.

2) Historical Data Access (HDA) схожа с DA, но позволяет получать от оборудования не текущие значения, а их историю или историю их изменений.

3) Alarms & Events (AE) предоставляет механизмы уведомлений о нештатных ситуациях на производстве: тревоги, отказ оборудования, потеря связи с наблюдаемым объектом и т.д.

Помимо основных спецификаций, OPC UA включает Batch (пакетное управления оборудованием), Data Exchange (обмен информацией через Ethernet),

XML-DA (SOAP и RPC реализация методов DA) и Security (управление доступом).

OPC UA SDK позволяет программистам разрабатывать универсальные приложения самостоятельно. Существуют реализации фреймворка для большинства распространённых языков, включая C++, C# и Java. Исходный код библиотек доступен на Github и распространяется под лицензией GPL. Кроме того, компании, являющиеся действующими членами OPC Foundation Corporate, могут использовать лицензию RCL. Лицензия позволяет использовать разработанное ПО в коммерческих целях без открытия его исходного кода. Сторонние компании также предлагают собственные решения на основе спецификаций OPC UA.

1.4 Unified Automation Expert Solutions

Unified Automation Expert [28] предлагает готовые OPC UA приложения для типовых случаев использования. Такие приложения могут использоваться без дополнительной настройки и доработки совместно с уже существующими клиентскими OPC UA и OPC Standard приложениями.

Компания также предлагает специализированный пакет для моделирования структуры нижележащих аппаратных систем – UaModeler (рисунок 3) [21].

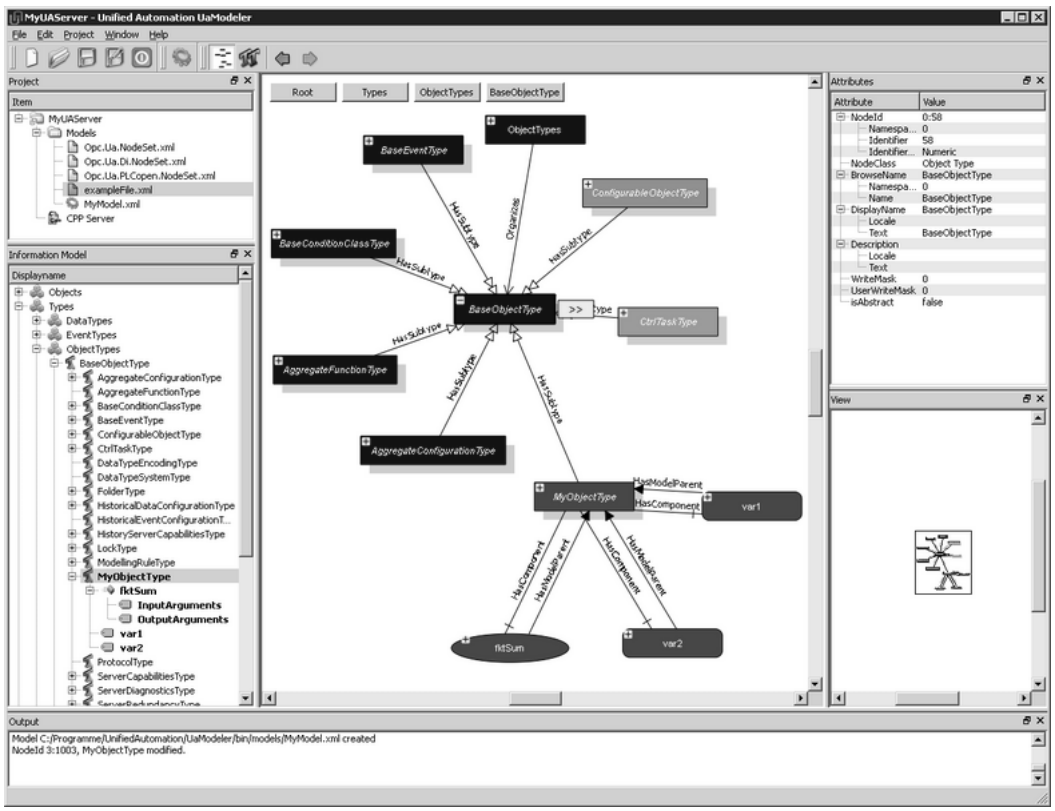


Рисунок 3 – Приложение UaModeler

UaModeler предоставляет графический интерфейс для создания модели приложения. Модель включает в себя создание адресного пространства, добавление узлов, настройку ссылок между ними и генерацию исходного кода. Код полностью совместим с официальной спецификацией OPC и OPC UA. Это ускоряет разработку и повышает качество новых решений. Сгенерированный код хорошо структурирован и не содержит логических ошибок. Благодаря подходу «Modeling & Generating», даже сложные многоуровневые модели могут быть созданы в короткий срок.

Для более сложных случаев UA Expert предлагает собственное SDK, основанное на OPC UA ANSI C Communication Stack. SDK доступен для систем под управлением Windows, Windows CE и Linux. Предоставление SDK для других операционных систем может быть организовано по запросу. .Net SDK доступно только для ОС Windows.

Таким образом, Unified Automation предлагает полный альтернативный стек для компаний, не желающих сотрудничать с OPC Foundation Corporate. Однако, использование стороннего программного стека влечет за собой характерные

проблемы: закрытый исходный код, слабая поддержка сообщества, необходимость приобретения дополнительных инструментов. SMS Group является официальным членом OPC Foundation Corporate. Компании выгоднее придерживаться официального стека разработки. Кроме того, SMS Group имеет собственный R&D отдел, который может вести разработку нового программного обеспечения самостоятельно.

1.5 Genius CM System

SMS Group GmbH (Германия) – международная компания, работающая в сфере металлургии и литья. Компания предоставляет оборудование и услуги для модернизации и дигитализации предприятий. SMS Group также ведет разработку программного обеспечения и проводит научные исследования, направленные на оптимизацию процессов литья. Одним из продуктов компании является система genius Condition Monitoring (Genius CM) [2].

Genius CM – программное обеспечение для мониторинга состояния аппаратного оборудования литейных машин. Система состоит из компонентов, отвечающих за анализ показателей, базы данных и Web-интерфейса. В простейшем случае все компоненты системы располагаются на одном компьютере, однако возможно использование распределенных компонентов анализа, отправляющих данные в центральную БД. Web-сервер также может быть развернут на удаленной машине.

Система обрабатывает данные, поступающие от IBA PDA станций, и анализирует показатели датчиков. Текущие показатели и история их значений отображается в Web-интерфейсе (рисунок 4) в виде графиков и/или таблиц. Genius CM позволяет указать, значения каких параметров и за какой промежуток времени необходимо отображать. Система также предоставляет уведомления о нештатных ситуациях и тревогах. Кроме того, Genius CM предоставляет Rest-API для управления компонентами анализа.

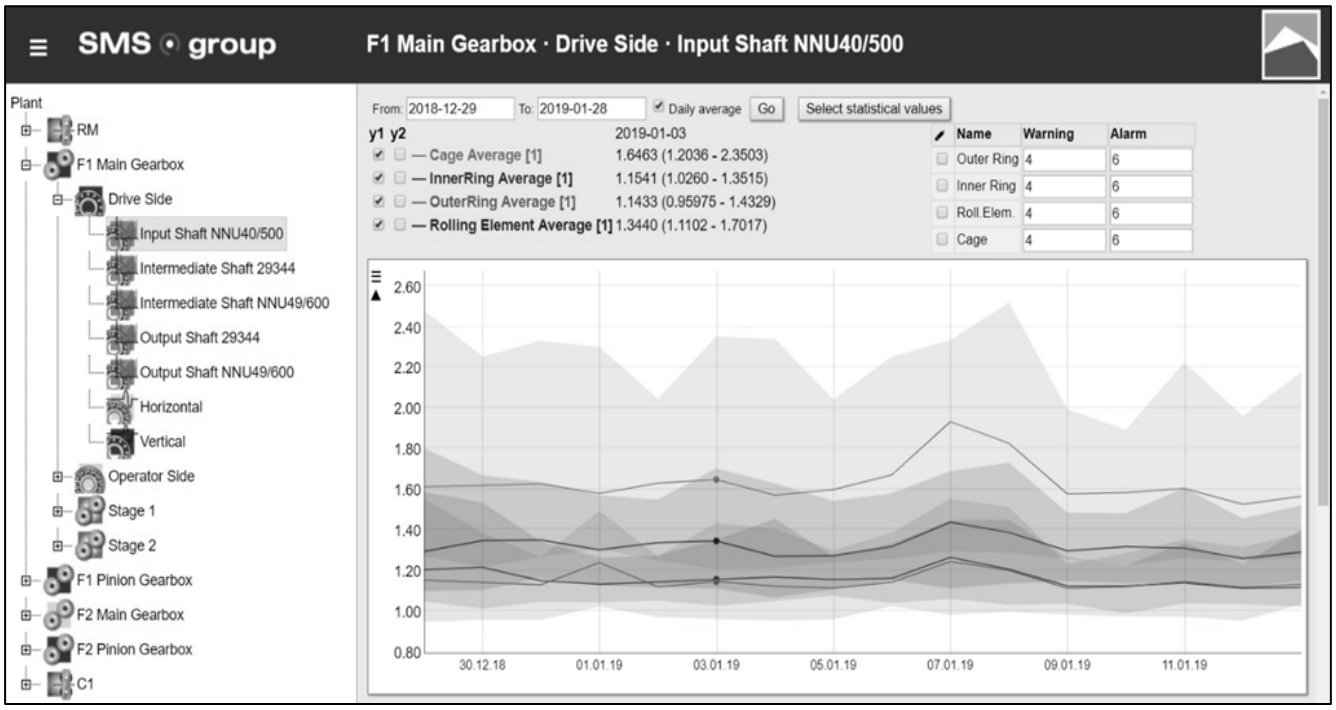


Рисунок 4 – Web-интерфейс системы Genius CM

Структура завода отображается в левой части интерфейса в виде иерархического дерева. Верхний уровень отображает комплексы оборудования: печи, прокатные станы, устройства охлаждения и т.д. Второй уровень содержит группы станков или этапов производства. Каждый лист дерева отражает станцию аналитики конкретного станка. В центральной части отображаются графики и таблицы значений, а также элементы управления.

Genius CM задумывалась как «Human-to-Machine» система. Она предоставляет крайне ограниченный API, не позволяющий запрашивать показатели сенсоров или информацию о тревогах. Пользователь может взаимодействовать с системой только через Web-браузер. Использование Genius CM в качестве M2M-приложения не представляется возможным.

Выводы по разделу

Постепенный переход на производственный стандарт Industry 4.0 требует модернизации предприятий и внедрения новых систем управления. Организация взаимодействия между устройствами промышленной автоматизации является критически важной задачей. Она делает возможным сбор и анализ информации на

каждом этапе производственного процесса, позволяет повысить эффективность производства, оперативно принимать бизнес-решения [23].

HMI-системы утрачивают свою важность, а на первое место выходят интерфейсы межмашинного взаимодействия, такие как OPC. Современная спецификация OPC UA лишена недостатков предыдущего стандарта OPC Classis. Она не привязана к операционной системе или языку программирования и широко поддерживается производителями оборудования. OPC UA является наиболее подходящим стандартом для создания M2M-приложения.

Использование готовых OPC-систем не представляется возможным; необходима разработка приложения под конкретную задачу. Разработка OPC UA сервера для системы Genius CM позволит значительно расширить ее возможности, сделать систему доступной для M2M-взаимодействия.

2 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

Проектирование ПО является важной частью процесса разработки. Цель проектирования – выявление внутренних и детализация внешних свойств приложения, основываясь на требованиях к системе. Основательный подход к проектированию позволяет достичь лучшего понимания задачи и избежать разногласий с заказчиком. Это приводит к значительному сокращению времени разработки продукта [19].

OPC UA сервер будет интегрирован с системой Genius CM. Система получает информацию от IBA PDA станций, установленных на заводском оборудовании. Данные обрабатываются и сохраняются в SQLite БД. Основной задачей сервера станет предоставление клиентам унифицированного доступа к ресурсам Genius CM. Общая схема работы системы представлена на рисунке 5.

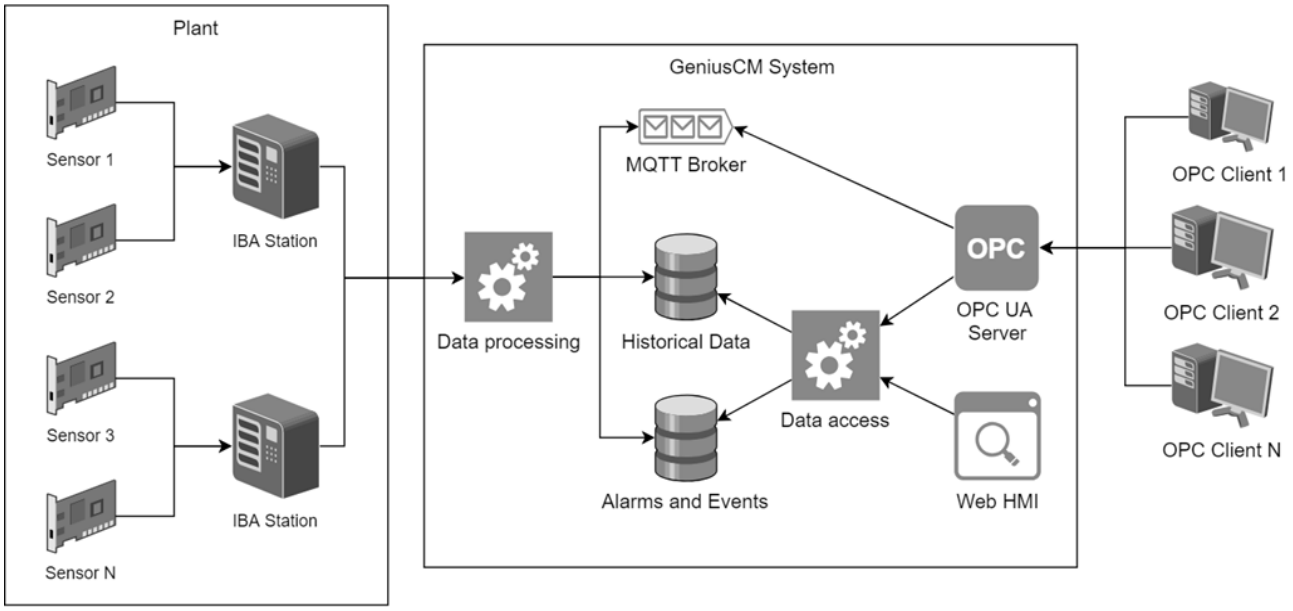


Рисунок 1 – Общая схема работы системы

Рассмотрим подробнее работу Genius CM. За обработку поступающих данных отвечает компонент DataProcessing. Данные анализируются, новые значения вычисляются на основе имеющихся. Показатели датчиков вместе и результаты анализа вместе с соответствующей меткой времени сохраняются в базу данных Historical Data. Эти данные могут быть запрошены позднее с помощью Web-интерфейса или HDA сервиса. При выявлении отклонения значения от нормы,

компонент создает запись в БД Alarms and Events. Компонент формирует сообщение, содержащее актуальные значения, и отправляет его через MQTT брокер.

OPC UA сервер получает актуальные данные от MQTT и использует компонент Data Access для доступа к истории показателей и данным о тревогах за определенный период. OPC UA сервер предоставляет M2M интерфейс взаимодействия с системой, в то время как Web-интерфейс служит для взаимодействия с людьми.

2.1 Требования к системе

В процессе проектирования сервера были проведены встречи с разработчиками системы Genius CM. Было составлено техническое задание на разработку OPC UA сервера, выявлены требования к системе.

Функциональные требования

Функциональные требования содержат информацию о том, как должна быть реализована система, и какие возможности она должна предоставлять. Они содержат бизнес-требования, которые определяют назначение ПО, и требования пользователя, определяющие набор пользовательских задач. Для разрабатываемой системы были выявлены следующие функциональные требования:

1) Сервер должен работать в соответствии с спецификациями OPC UA. Он должен поддерживать сервисы DA, HDA и AE.

2) Авторизация пользователя должна производиться на основе логина и пароля. Анонимный доступ не допускается. Авторизованный пользователь должен получать доступ ко всем возможностям сервера.

3) Сервер должен отображать аппаратное оборудование завода в виде иерархического дерева. Уровни дерева должны иметь такой же смысл, как и в Web-интерфейсе Genius CM.

4) Сервер должен предоставлять пользователям текущие показатели датчиков и результаты анализа в режиме реального времени. Сервер должен корректно обрабатывать случаи, когда часть данных повреждена или отсутствует.

5) Сервер должен предоставлять доступ к истории наблюдаемых значений за указанный период времени. В случае, если запрашиваемые данные отсутствуют, сервер должен возвращать соответствующий код ошибки.

6) Сервер должен создавать уведомления о нештатных ситуациях и тревогах на производстве. Тревоги должны разделяться по уровню срочности, предоставлять информацию об источнике и времени возникновения.

7) Сервер должен предоставлять методы для управления станциями аналитики. Должны поддерживаться запрос текущего статуса станции, ее включение и отключение.

Нефункциональные требования

Нефункциональные требования определяют критерии работы системы в целом, а не отдельные аспекты ее поведения. Нефункциональные требования определяют такие свойства как производительность, удобство сопровождения, расширяемость и отказоустойчивость системы. OPC UA сервер должен соответствовать следующим нефункциональным требованиям:

1) Реализация должна быть выполнена на языке C# 7.0 [25] с использованием .Net Framework 4.6 или .Net Standard 1.3 [1].

2) Сервер должен корректно работать на ПК под управлением ОС Windows версий 7 SP 1, 8, 8.1 и10.

3) Время ответа на запрос не должно превышать 3 сек. В случае, если сервер не имеет возможности обработать запрос за указанное время, он должен вернуть соответствующий код ошибки.

4) Сервер должен поддерживать работу с несколькими клиентами.

5) Сервер должен взаимодействовать с системой Genius CM по установленному протоколу. Получение онлайн-данных должно осуществляться с помощью MQTT брокера. Для управления станциями анализа должны использоваться Web-API методы Genius CM.

6) Сервер должен осуществлять логирование своей работы. Логи должны храниться в TXT формате.

7) Сохранение конфигурации сервера должно осуществляться в XML или INI формате.

2.2 Описание прецедентов

В качестве пользователя системы может выступать другое приложение, работающие в соответствии с спецификацией OPC UA. Взаимодействие клиента с системой продемонстрировано на диаграмме вариантов использования (рисунок 6).

Рассмотрим подробнее пользовательские прецеденты [26].

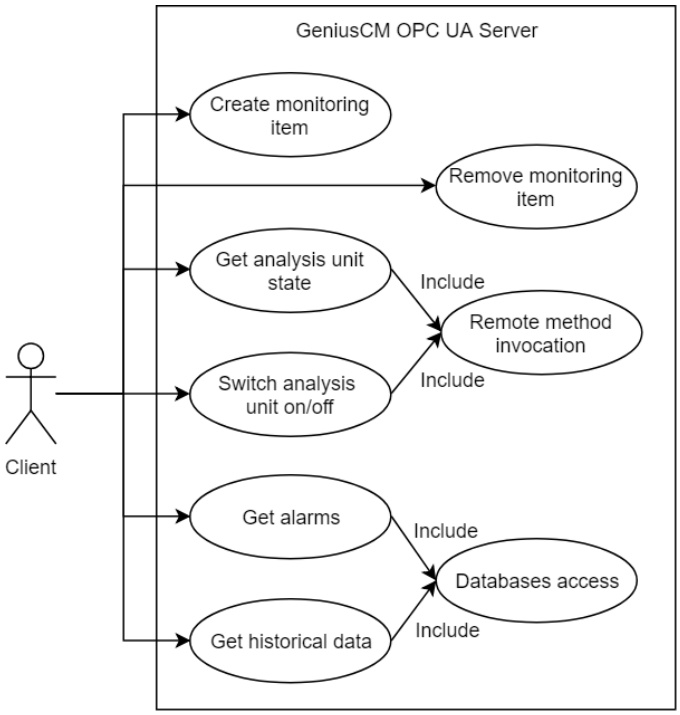


Рисунок 2 – Диаграмма вариантов использования

1) *Create monitoring item* – создание нового наблюдаемого значения. Клиент запрашивает текущее значение определенного параметра и подписывается на его обновления. Клиент может запросить наблюдение сразу за несколькими параметрами, принадлежащими разным станциям анализа.

2) *Remove monitoring item* – удаление наблюдаемого значения. Клиент сообщает о аннулировании подписки на обновления определенного параметра. Сервер прекращает отправку сообщений о обновлении данного параметра.

3) *Get analysis unit state* – запрос текущего состояния указанной станции анализа.

4) *Switch analysis unit on/off* – запрос на включение/отключение указанной станции анализа.

5) *Get alarms* – запрос имеющихся данных о нештатных ситуациях.

6) *Get historical data* – запрос истории значений указанного параметра за определенный промежуток времени. Сервер находит запрошенные данные в БД и отправляет клиенту.

7) *Remote method invocation* – сервер осуществляет вызов метода Genius CM Web API.

8) *Database access* – сервер осуществляет поиск значений в базе данных Genius CM.

2.3 Доменная модель

Доменная модель содержит описание объектов предметной области, относящейся к программе. Построение доменной модели позволяет определить основные сущности приложения без их детального описания и реализации [29].

Доменная модель OPC UA сервера (рисунок 7) имеет три уровня. Рассмотрим их более подробно.

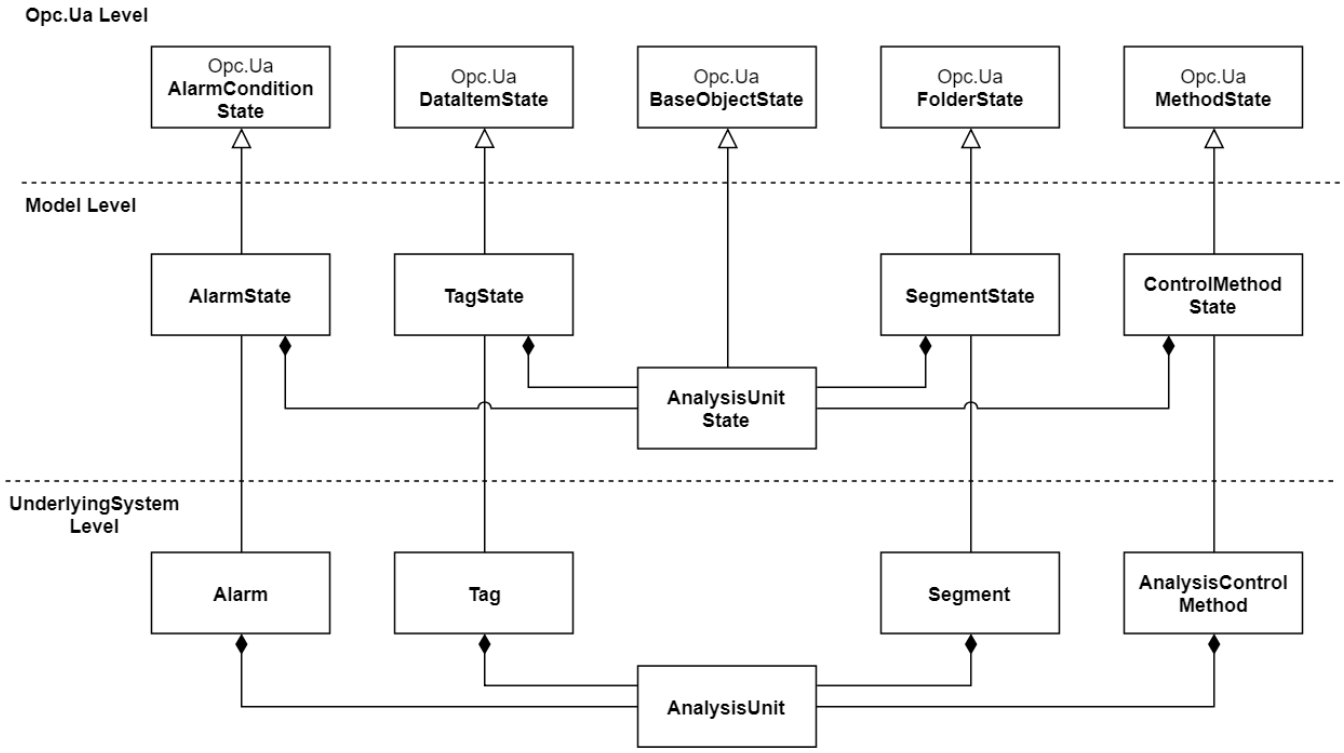


Рисунок 3 – Доменная модель приложения

Аппаратная система

Первый уровень модели описывает объекты аппаратной системы завода: станции анализа и связанные с ними сущности.

AnalysisUnit – представляет станцию анализа, которая является основным объектом аппаратной системы. Каждый литейный аппарат оборудован несколькими станциями наблюдения, расположенными на разных подсистемах, поэтому станции объединяются в сегменты.

Segment – представляет директорию с дочерними элементами. Такими элементами могут быть станции анализа, теги или другие директории. Литейный аппарат также представляется как директория, так как он не имеет дополнительных свойств или поведения.

Tag – описывает параметр датчика или станции анализа. Тег предоставляет доступ к обновляющемуся значению параметра и истории этих значений. Список тегов, поддерживаемых станцией, задается в файле конфигурации.

Alarm – сущность содержит информацию о тревоге или нештатной ситуации: время возникновения, причина, срочность, зафиксировавшая станция наблюдения и т.п. Также сущность отвечает за подтверждение и отмену тревоги.

AnalysisControlMethod – определяет метод управления станцией наблюдения. Каждая станция имеет методы для включения/отключения, запроса состояния и т.п. Метод содержит название, описание, входные аргументы и результат.

Модель

Второй уровень содержит вспомогательные сущности, являющиеся «мостом» между аппаратной системой и спецификаций OPC. Каждая из сущностей уровня модели является наследником стандартного OPC-объекта и расширяет его поведение.

AnalysisUnitState – инкапсулирует станцию наблюдения, представляя ее как OPC BaseObject. Модель предоставляет отдельные сегменты для тегов, свойств и методов станции.

TagState – выполняет преобразование тега к OPC DataItem [15]. Модель хранит последнее актуальное значение тега и отвечает за его обновление.

SegmentState – представляет сегмент в виде OPC FolderState, который может хранить любые другие OPC State и ссылки на дочерние элементы.

AlarmState – преобразует сущность Alarm в OPC AlarmConditionState [17]. Модель отвечает за жизненный цикл тревоги: возникновение, обновление важности, подтверждение и закрытие.

OPC UA Objects

Верхний уровень доменной модели содержит сущности OPC UA. Эти объекты реализованы в официальном фреймворке OPC UA .Net в соответствии с спецификацией и могут использоваться любым OPC UA клиентом.

BaseObjectState – представляет базовый объект OPC. Все остальные OPC-состояния унаследованы от него. Объект не указывает дополнительного поведения или свойств, однако может содержать подпапки и теги.

FolderState – представляет директорию в OPC UA. Экземпляры этого типа используются для организации адресного пространства сервера в виде иерархии каталогов. FolderState определяет узел с именем и дочерними элементами.

DataItemState – является базовым объектом для OPC переменных. Переменная с таким типом может хранить значение любого OPC или пользовательского типа.

PropertyState – представляет стандартное свойство объекта, является подтипом DataItem. Свойство имеет имя и хранит заранее заданное значение. В отличии от переменной, значение свойства, как правило, не меняются либо меняются крайне редко.

Выводы по разделу

Планирование архитектуры является важной частью процесса разработки: хорошо проработанная архитектура уменьшает время разработки ПО, облегчает будущую поддержку приложения, а также способствует доработке функционала. Кроме того, для построения приложения необходим подробный анализ пользовательских требований.

В этой главе была разработана архитектура OPC UA приложения. Были достигнуты следующие результаты:

1) Рассмотрены функциональные и нефункциональные требования.

2) Описаны возможные варианты использования; создана диаграмме пользовательских прецедентов.

3) Были выделены основные компоненты и объекты приложения, описаны взаимосвязи между ними; построена доменная модель.

На основе разработанной архитектуры может быть начата реализация OPC UA сервера для системы Genius CM.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

В соответствии с установленными требованиями, разработка велась на языке C# 6.0. Приложение использует .Net Framework v4.6. Эта версия .Net реализует .Net Standard v1.3 и обеспечивает хороший баланс между доступными функциями и обратной совместимостью [25].

Для реализации спецификаций OPC UA был использован официальный стек OPC UA .Net Standard Stack от OPC Foundation. Фреймворк предоставляет эталонную реализацию OPC UA для .NET Standard [18]. Он позволяет разрабатывать универсальные приложения, работающие на большинстве платформ, распространенных на сегодняшний: Linux, iOS, Android (через Xamarin) и Windows 7/8 /8.1/10 (включая встраиваемые и IoT-версии). Модификации кода для конкретной платформы не требуются. Основные преимущества стандартного стека:

- Полностью портированное ядро UA Core и SDK (клиент, сервер, средства конфигурации);
- Примеры реализации серверов, клиентов и элементов управления для .Net 4.6, .NetCore 2.0 и UWP;
- Поддержка SHA-2 (вплоть до SHA512). Профиль безопасности Basic256Sha256 для систем с высокими требованиями к безопасности;
- Официальные Nuget-пакеты со сборками ядра, серверных компонентов и средств конфигурации. Простоя интеграция с .Net проектами;
- Сертификация о соответствии стандартам OPC. Стек был протестирован в лаборатории OPC Foundation с помощью инструмента тестирования соответствия (CTT).

3.1 Адресное пространство и узлы в OPC UA

Набор объектов и связанной с ними информации, которую сервер предоставляет клиентам, называется адресным пространством сервера [13]. Его основная задача – предоставление клиентам доступа к объектам сервера.

Объектная модель OPC UA (рисунок 8) была разработана для достижения этой цели. Подобно объекту в ООП парадигме, OPC-объект может содержать набор переменных и методов, а также порождать события. Между собой объекты связываются ссылками.

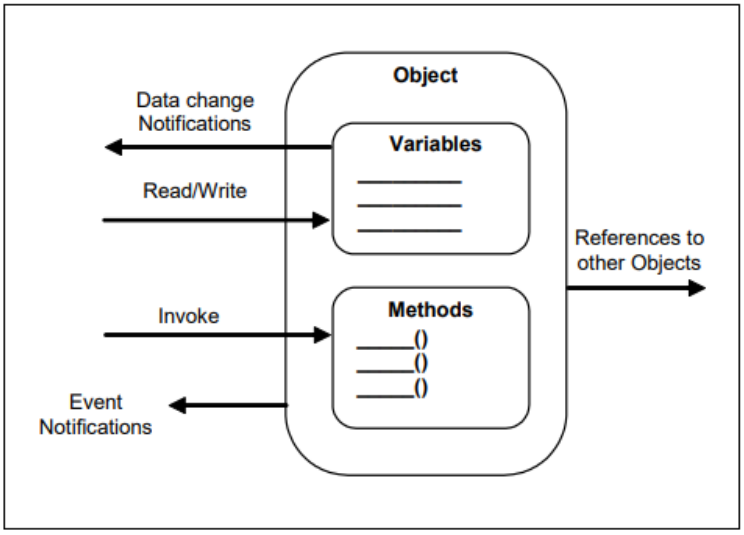


Рисунок 4 – Объектная модель OPC UA

Элементы объектной модели представляются в адресном пространстве в виде узлов (*Nodes*) [11, 14]. Каждому узлу присваивается класс (*NodeClass*), который определяет конкретный элемент модели. Узел имеет набор обязательных атрибутов. Например, уникальный идентификатор (*NodeId*) и отображаемое имя. Ссылки на другие узлы необязательны, но, если узел не связан ни с одним другим узлом, он оказывается недоступен: клиент не сможет найти его в адресном пространстве.

Атрибут – это элемент данных, описывающий узел. Клиенты могут получать и изменять значения атрибутов с помощью методов Read, Write и Query. Кроме того, клиент может подписаться на изменения значения атрибута с помощью сервиса MonitoringItem. Тип атрибута является частью NodeClass, однако не включается в адресное пространство. Тип включает в себя идентификатор, название, описание, тип данных и битовый маркер, определяющий, является ли атрибут обязательным. Значения атрибутов вычисляются при добавлении узла в адресное пространство.

Ссылки используются для связи узлов между собой. Доступ к ним можно получить с помощью сервисов Browse и Query. Подобно атрибутам, ссылки являются неотъемлемой частью NodeClass, но доступны в адресном пространстве. Тип ссылки определяет взаимоотношение между узлами. Узел, который содержит ссылку, называется исходным узлом (*source node*), а узел, на который она указывает – целевым (*target node*). Ссылка должна быть уникальной: OPC не допускает создание нескольких ссылок одинакового типа, имеющих совпадающие исходные и целевые узлы.

Структура узла с ссылками и атрибутами представлена на рисунке 9.

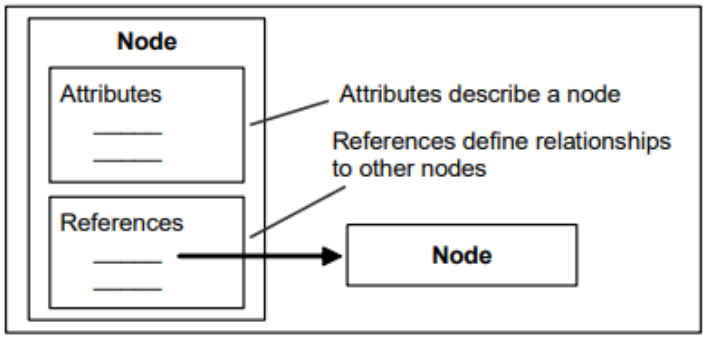


Рисунок 5 – Атрибуты и ссылки узла

3.2 Менеджер узлов

Класс, отвечающий за создание адресного пространства сервера, называется менеджером узлов (*Node Manager*) [22]. Такой класс должен реализовать интерфейс INodeManager (или INodeManager2 в новых версиях) и INodeIdFactory, предоставленные OPC UA .Net Standard Stack. Интерфейс INodeManager содержит сигнатуры методов, отвечающих за CRUD-операции адресного пространства, доступ к онлайн и историческим данным, обработку событий и методов обратного вызова (*callback*). INodeIdFactory содержит только один метод New, который должен возвращать уникальный NodeId для каждого нового узла.

Фреймворк предоставляет реализацию CustomNodeManager (рисунок 10), который может использоваться как базовый при разработке менеджера узлов приложения. Класс содержит более 40 вспомогательных методов, доступных для переопределения.

Например, интерфейс INodeManager подразумевает только один метод для обработки всех типов запросов к историческим данным – HistoryRead. Вместо этого, CustomNodeManager предоставляет 4 метода: HistoryReadRaw для запроса «сырых» данных, HistoryReadProcessed для чтения агрегированных или предобработанных значений, HistoryReadAtTime для чтения значений с определенными меткам времени.

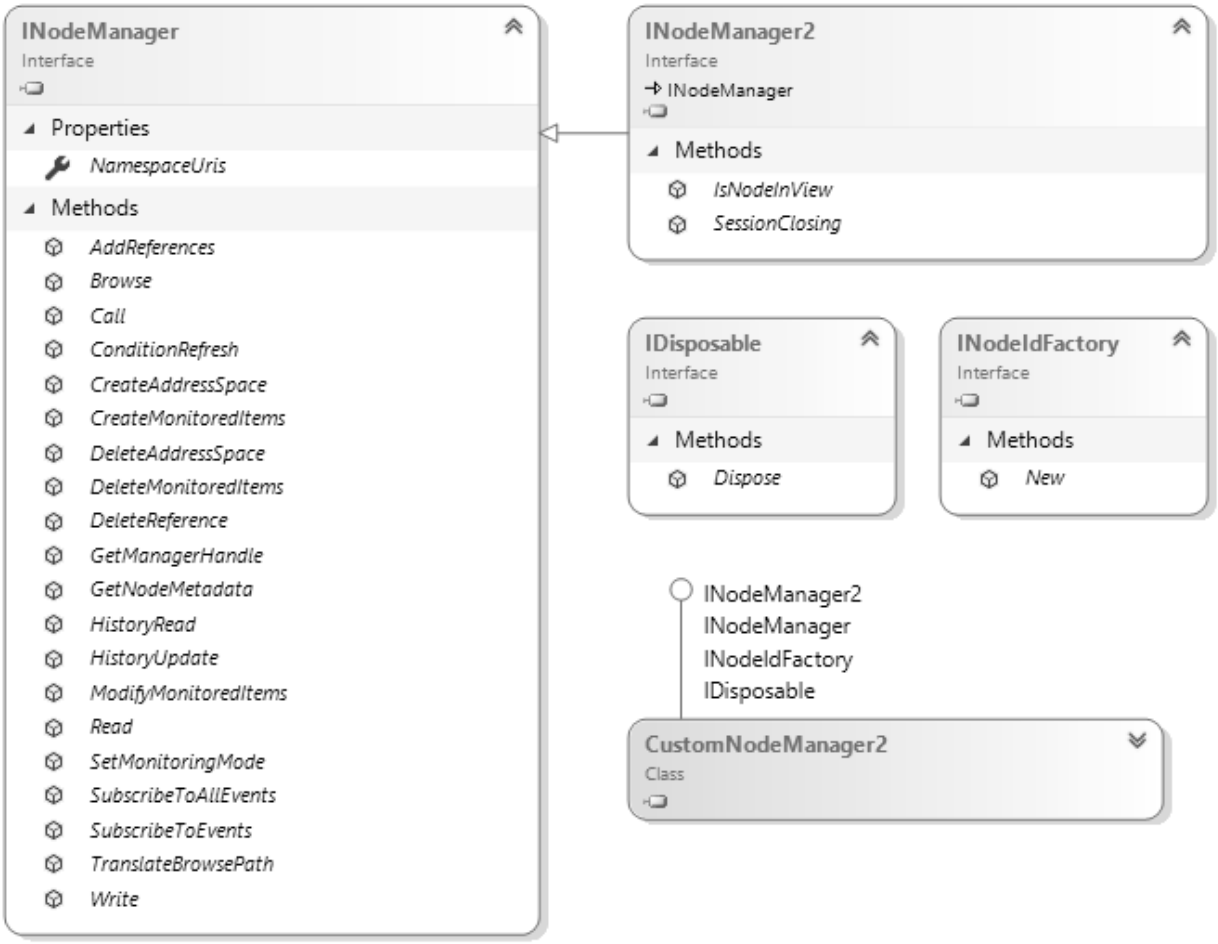


Рисунок 6 – Интерфейс INodeManager и класс CustomNodeManager

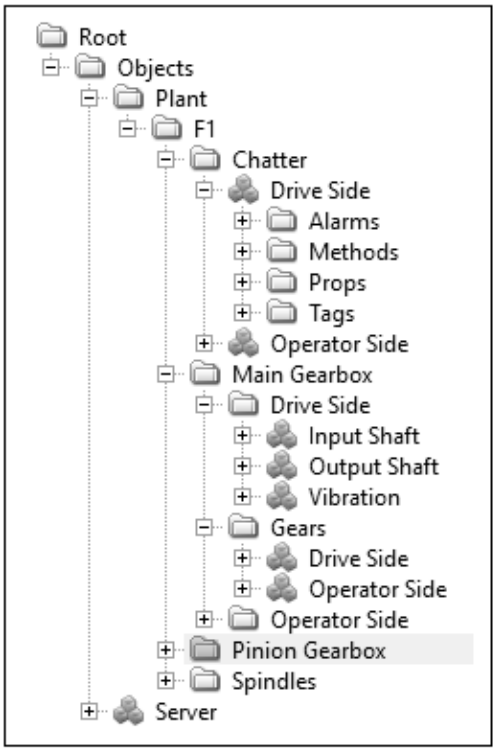
Менеджер узлов является входной точной для клиентов: сервер проксирует запросы клиента соответствующему менеджеру. Для OPC UA сервера был создан GeniusNodeManager. Класс наследует реализацию CustomNodeManager и переопределяет его методы доступа к данным. Рассмотрим его реализацию более подробно.

Для создания менеджера требуется XML файл с конфигурацией аппаратной системы завода (приложение А). Конфигурация имеет древовидную структу-

ру. Каждый тег *PlantNode* описывает определенное аппаратное устройство. Элемент имеет название и уникальный идентификатор. Он может содержать дочерние элементы или описание станции анализа (тег *Analysis*). Станция анализа имеет идентификатор, название и класс. Класс определяет список параметров, которые может предоставлять станция. Списки таких значений хранятся в ещё одном XML файле (приложение Б). Тег *column* описывает параметр: название, группа и единицы измерения. Атрибут *show* необходим для веб-интерфейса Genius CM и игнорируется OPC UA сервером. Файл содержит описание 23 классов анализа с более чем 450 параметрами суммарно.

GeniusNodeManager создает корневой сегмент Plant, в который помещаются все объекты в соответствии с их иерархией. Директория добавляется в Objects – стандартную директорию для публичных объектов OPC сервера.

Пример полученного адресного пространства приведен на рисунке 11 (а). Тестовая конфигурация включает один прокатный стан F1, состоящий из 4 сегментов: Chatter, Main Gearbox, Pinion Gearbox и Spindles. Каждый сегмент имеет несколько станций аналитики. Станции создаются с подпапками для тревог, методов, свойств и тегов. Пример списка тегов приведен на рисунке 11 (б).



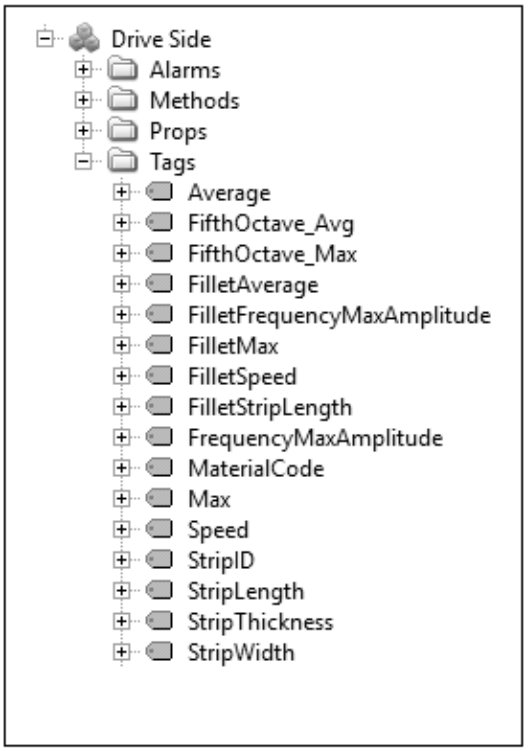


Рисунок 7 – Адресное пространство (а) и список тегов (б)

3.3 Data Access Service

Задачей DA сервиса является предоставление доступа к данным автоматизации. Данные могут располагаться непосредственно на сервере или на устройствах ввода-вывода, подключенных к нему. Они также может быть запрошены у вспомогательных серверах или других устройств, таких как аппаратные контроллеры. DA сервис должен предоставлять OPC UA клиентам прозрачный доступ к своим данным автоматизации. Какие категории данных и в каком виде будут представляться целиком зависит от сервера и производителя оборудования [16].

Классы, представляющие данные автоматизации, называются элементами данных (*DataItems*). Клиенты могут читать или записывать значения таких элементов или отслеживать их изменения. DataItem не может существовать в адресном пространстве самостоятельно, он должен быть компонентом какого-либо узла. Каждый элемент данных унаследован от типа DataVariable и имеет определенный набор обязательных атрибутов. Рассмотрим наиболее важные из них:

- *Value* – актуальное значение элемента
- *DataType* – тип данных. OPC UA предоставляет множество стандартных типов данных, включая char, string, integer, real, bool, binary и image. Поддерживаются и более специфичные типы. Например, географические координаты и комплексные числа.
- *AccessLevel* – определяет уровень доступа к значению. Позволяет создать read-only элементы.
- *ValueRank* – задает число измерений. Ноль – скаляр, один – массив, два – матрица и т.п.

Дополнительные характеристики элемента данных могут быть заданы с помощью свойств.

Реализация

Рассмотрим подробнее реализацию DA сервиса. Для поддержки доступа к данным, менеджер узлов должен переопределить реализацию методов OnMonitoredItemCreated и OnMonitoredItemDeleted. За вызов отвечает код ядра Core UA.

Методы вызываются при создании и удалении DataItem соответственно. Если клиент запрашивает создание или удаление сразу нескольких элементов, соответствующий метод будет вызван для каждого из них. Методы имеют одинаковую сигнатуру, которая представлена на рисунке 12.

```
protected virtual void OnMonitoredItemCreated(ServerSystemContext context, NodeHandle handle, MonitoredItem monitoredItem)

protected virtual void OnMonitoredItemDeleted(ServerSystemContext context, NodeHandle handle, MonitoredItem monitoredItem)
```

Рисунок 8 – Методы OnMonitoredItemCreated и OnMonitoredItemDeleted

Методы ожидают три входных аргумента. *Context* содержит контекст текущей сессии. Он необходим, так как один сервер может работать с несколькими клиентами параллельно, и не может использовать глобальные переменные для хранения состояния. *Handle* содержит ссылку на узел, для которого вызван метод. *MonitoredItem* – объект мониторинга – созданный сервером объект, который будет отправлен клиенту. Позволяет настроить параметры и частоту обновления наблюдаемого значения.

Сервер управляет созданием клиентских подписок. По умолчанию, одна подписка создается для одного клиента. Если клиент запрашивает наблюдение за новым объектом мониторинга, он добавляется в существующую подписку. Сервер публикует обновления несколько раз в секунду. После удаления всех MonitoredItem или отключения клиента, подписка уничтожается.

OPC UA позволяет переопределить управление подписками, если это необходимо. Сервер может создать несколько подписок для одного клиента или даже отдельную подписку для каждого объекта мониторинга. Это позволяет более гибко настроить публикацию обновлений. Например, возможно разделить подписки по приоритету и публиковать обновления чаще для более важных объектов.

Тем не менее, создание и обновление подписки является достаточно ресурсоемкой операцией. Создание большого числа подписок может негативно сказаться на производительности сервера.

Онлайн данные поступают от системы Genius CM через MQTT брокер. Протокол MQTT [9] нацелен на простоту в использовании, низкую загрузку канала передачи данных и простую интеграцию в систему. Протокол не накладывает ограничения на тип или формат передаваемых данных. Использование паттерна «издатель-подписчик» позволяет новым клиентам получать информацию об обновлениях, не устанавливая предварительный контакт с Genius CM.

Как только система получает новые данные от станций наблюдения, она публикует широковещательное сообщение. Сообщение содержит id станции и обновленные значения параметров в формате JSON.

OPC UA сервер подписывается но получение сообщений при запуске. Если сообщение относится к станции, которая находится под наблюдением хотя бы одного клиента, сервер обновляет значения тегов. Рассмотрим фрагмент кода класса AnalysisUnit, который выполняет обновление тегов (рисунок 13).

Класс содержит делегат TagsChangedEventHandler и его экземпляр – поле onTagsChanged. Делегат описывает метод с одним входным аргументом – списком обновленных тегов.

Метод StartMonitoring вызывается при первом запросе наблюдения за одним из тегов станции. Он создает обработчик широковещательных сообщений от Genius CM и сохраняет ссылку на функцию обратного вызова, которая вызывается каждый раз, после обновления значений.

OnDataMessageReceived вызывается при получении нового MQTT сообщения. Если оно относится к другой станции или отсутствует метод обратного вызова, сообщение игнорируется. Иначе выполняется обновление тегов. Поле Tags содержит список поддерживаемых станцией тегов, сообщение – словарь Values, позволяющий получить обновленное значение тега по его имени. Если, по какой-либо причине, новое значение отсутствует, статус код тега выставляется в

BadNoDataAvailable. Теги добавляются в список, который затем передается в функцию обратного вызова. Исходный код OnTagsChanged представлен на рисунке 14.

```
public class AnalysisUnit
{

  public delegate void TagsChangedEventHandler(IList<Tag> updatedTags);
  private TagsChangedEventHandler _onTagsChanged;

  public void StartMonitoring(TagsChangedEventHandler callback)
  {
    _onTagsChanged = callback;
    MqttDataProvider.OnMessageReceived += OnDataMessageReceived;
  }

  private void OnDataMessageReceived(Message message)
  {
    if (message.AnalysisClass != _analysisClass ||
        _onTagsChanged == null)
      return;

    var updatedTags = new List<Tag>();
    foreach (var tag in Tags)
    {
      if (message.Values.TryGetValue(tag.Name, out var newValue))
      {
        tag.Value = newValue;
        tag.StatusCode = StatusCodes.Good;
      }
      else {
        tag.StatusCode = StatusCodes.BadNoDataAvailable;
      }
      tag.Timestamp = message.StartTime;
      updatedTags.Add(tag);
    }

    _onTagsChanged?.Invoke(updatedTags);
  }
}
```

Рисунок 9 – Исходный код класса AnalysisUnit

```
void OnTagsChanged(IList<Tag> updatedTags) {
  lock (_nodeManager.Lock) {
    foreach (var tag in updatedTags) {
      if (TagsSegment.FindChildBySymbolicName(
          _nodeManager.SystemContext, tag.Name) is TagState tagState) {
        tagState.StatusCode = tag.StatusCode;
        tagState.Value = tag.Value;
        tagState.Timestamp = tagState.Timestamp;
      }
    }

    ClearChangeMasks(_nodeManager.SystemContext, true);
   }
}
```

Рисунок 10 – Исходный код OnTagsChanged

Для корректной работы в многопоточной среде все действия с OPC State-объектами должны выполняться в критической секции. Обработчик получает блокировку на lock-объекте менеджера узлов. Затем производится поиск DataItem по имени тега. Если элемент найдет, он получает новое значение, метку времени и статус. После обновления всех элементов вызывается метод ClearChangeMasks. Этот метод сохраняет новое состояние объекта и всех его дочерних элементов.

3.4 Historical Data Access Service

HDA сервис предоставляет клиентам доступ к различным историческим данным и источникам исторических событий [12]. Данные могут быть расположены в БД или в памяти. Сервис может вести историю для всех или подмножества доступных переменных, объектов или свойств в адресном пространстве сервера. HDA может быть реализован в виде автономного OPC UA сервера, который собирает данные другого сервера или другого источника данных.

Модель HDA определяет дополнительные типы узлов, такие как *HistoricalDataNode* и *HistoricalEventNode*, а также объект аннотаций, который может присутствовать в HistoricalDataNode.

Аннотации определяют несколько важных свойств узла:

- *Historizing* – логическое свойство, отражающее, ведется ли запись истории для этого узла;

- *StartOfArchive* и *EndOfArchive* – содержат метки времени самой первой и самой последней доступной записи;
- *MinTimeInterval* и *MaxTimeInterval* – определяют размер минимального и максимального временного промежутка, доступного для чтения за 1 раз;
- *MinimumSamplingInterval* – задает минимальный промежуток времени между двумя сохраненными значениями.

Все эти свойства не являются обязательными, однако широко используются клиентскими приложениями. Например, временные границы архива помогают клиенту верно выбрать масштаб при отображении значений.

Реализация

Реализация HDA сервиса является более сложной задачей, по сравнению с реализацией DA. В первую очередь это связано с тем, что клиент может запросить историю разных типов. История может быть запрошена в сыром или предварительно обработанном виде. Например, вместо всех значений температуры может быть запрошено среднее значение по часам или по дням. Кроме того, могут быть запрошены значения для конкретных временных отметок.

Чтение большого фрагмента истории может потребовать большого количества времени и ресурсов сервера, поэтому OPC UA поддерживает кэширование запросов. Запрос может быть остановлен и сохранен в промежуточном состоянии в кэше с помощью *ContinuationPoint*. Обработка может быть продолжена из сохраненного состояния позже.

Рассмотрим реализацию простейшего случая доступа к истории: чтение «сырых» значений за определенный промежуток времени. Для этого менеджер узлов должен предоставить реализацию HistoryReadRawModified. Метод вызывается в двух случаях: при чтении необработанных исторических данных или при чтении истории изменений этих данных. Сигнатура метода приведена на рисунке 15.

```
protected virtual void HistoryReadRawModified(
      ServerSystemContext context,
      ReadRawModifiedDetails details,
      TimestampsToReturn timestampsToReturn,
      IList<HistoryReadValueId> nodesToRead,
      IList<HistoryReadResult> results,
      IList<ServiceResult> errors,
      List<NodeHandle> nodesToProcess,
      IDictionary<NodeId, NodeState> cache)
```

Рисунок 11 – Метод HistoryReadRawModified

Метод имеет 8 аргументов. Рассмотрим их более подробно:

1. *Context* – контекст текущей сессии.

2. *Details* содержит тип запроса (чтение истории значений или их изменений) и его дополнительные параметры. Например, максимальное число возвращаемых элементов, нужно ли возвращать пограничные значения, т.п.

3. *TimestampsToReturn* определяет, чьи (клиента или сервера) метки времени использовать, при выборке значений.

4. *NodesToRead* содержит узлы, которые необходимо прочитать.

5. *Results* должен содержать результаты выполнения запроса. Каждый элемент списка имеет поле HistoryData, которому должна быть присвоена история значений.

6. *Errors* должен содержать статус коды для каждого запрошенного тега. По умолчанию используется код BadNoData, означающий, что для указанного тега история отсутствует. Если запрос был обработан успешно, необходимо использовать статус код Good.

7. *NodesToProcess* содержит дескрипторы узлов, историю которых необходимо прочитать.

8. *Cache* может содержать кэшированные данные, если запрос был возобновлен после остановки или выполняется не впервые.

В отличии от методов DA, HistoryReadRawModified вызывается один раз для всех запрошенных данных. Причиной является то, что данные для разных тегов могут храниться в одной БД, и оптимальнее будет выбрать все запрошенные данные за один раз. Исходный код метода HistoryReadRawModified приведен на

рисунке 16.

Метод последовательно обрабатывает каждый запрос. Сначала он находит дескриптор и объект, в который будет записан результат. Затем дескриптор проверяется: если он указывает на несуществующий узел, запрос прерывается.

Дескриптор позволяет получить ссылку на узел, для которого выполняется запрос. Таким узлом должен быть TagState, который обеспечивает доступ к нижележащему тегу. Производится чтение истории тега. Метод TryReadHistory возвращает список пар «timestamp – значение». Полученные данные оборачиваются в объект HistoryData и сохраняются в список результатов.

Чтение истории изменений не поддерживается сервером, при запросе акого типа возвращается код BadNotSupported.

```
foreach (var handle in nodesToProcess) {
  HistoryReadValueId nodeToRead = nodesToRead[handle.Index];
  HistoryReadResult result = results[handle.Index];

  NodeState source = ValidateNode(context, handle, cache);
  if (source == null) {
    continue;
  }

  if (details.IsReadModified) {
    errors[handle.Index] = StatusCodes.BadNotSupported;
    continue;
  }

  var itemState = handle.Node as TagState;
  var tag = itemState?.Tag;
  if (tag == null) {
    continue;
  }

  if (!tag.TryReadHistory(details.StartTime, details.EndTime,
       out var data)) {
    errors[handle.Index] = StatusCodes.BadNoDataAvailable;
    continue;
  }

  var historicalData = new HistoryData();
  foreach (var valuePair in data) {
    var value = new Variant(valuePair.Value);
    var dataValue = new DataValue(value, StatusCodes.Good, valuePair.Key);
    historicalData.DataValues.Add(dataValue);
  }

  errors[handle.Index] = ServiceResult.Good;
  result.HistoryData = new ExtensionObject(historicalData);
}
```

Рисунок 12 – Реализация метода HistoryReadRawModified

3.5. Alarms and Events Service

Сервис AE построен на модели состояний, которые используются для представления состояния системы или одного из ее компонентов. Примерами таких состояний могут быть: выход температуры за определенный предел, нуждающееся в обслуживании устройство, требование подтвердить какой-либо шаг перед продолжением процесса, и т.д. Все состояния являются наследниками *ConditionType* [17]. Сервер решает, какие экземпляры состояний создавать, и будут ли они отображаться в адресном пространстве.

Базовая модель определяет только два состояния: отключено и включено. Переход между состояниями генерирует соответствующее событие, которое будет доставлено клиентам. Предполагается, что различные подтипы ConditionType, определенные в стандарте OPC UA, могут быть дополнительны и/или расширены поставщиками оборудования или другими группами стандартизации.

Одним из наследников ConditionType является *AcknowledgmentConditionsType*. Тип добавляет свойство Acknowledge, отражающее, было ли состояние подтверждено. Переход в подтвержденное состояние может осуществляться по запросу клиента или в результате работы внутренней логики сервера.

Модель «Acknowledge – Confirm» используется для разграничения между возникновением тревоги и выполнением действий для её устранения. Например, оператор получает предупреждение о высокой температуре двигателя одного из станков. Сначала он проверяет информацию, затем, если тревога подтвердилась, вызывает метод Acknowledge. Затем оператор предпринимает действия, чтобы вернуть температуру к нормальным значениям. После этого, вызывается метод Confirm, который сообщает серверу, что были предприняты корректирующие действия. Такая двухступенчатая система подтверждения аварийных сигналов может быть расширена для более сложных ситуаций [8].

Классы, описывающие аварийную ситуацию, являются наследниками AcknowledgeableCondition. Кроме описанных выше методов, они поддерживают методы Suppress (принудительное подавление тревоги) и Shelve (скрытие тревоги из зоны видимости). Если поле State такого класса имеет значение Active, это означает, что аварийная ситуация присутствует в данный момент. Значение Inactive сигнализирует о том, что показатели вернулись в нормальное состояние.

Типовой жизненный цикл тревоги показан на рисунке 17. Цель AE сервиса состоит в том, чтобы своевременно проинформировать оператора об опасных изменениях в процессе и позволить ему предпринять необходимые действия прежде, чем наступят какие-либо негативные последствия. Как правило, если в течение

некоторого периода времени не предпринимается никаких действий, связанных с аварийным сигналом, процесс пересекает некоторый порог, после которого начинают возникать последствия.

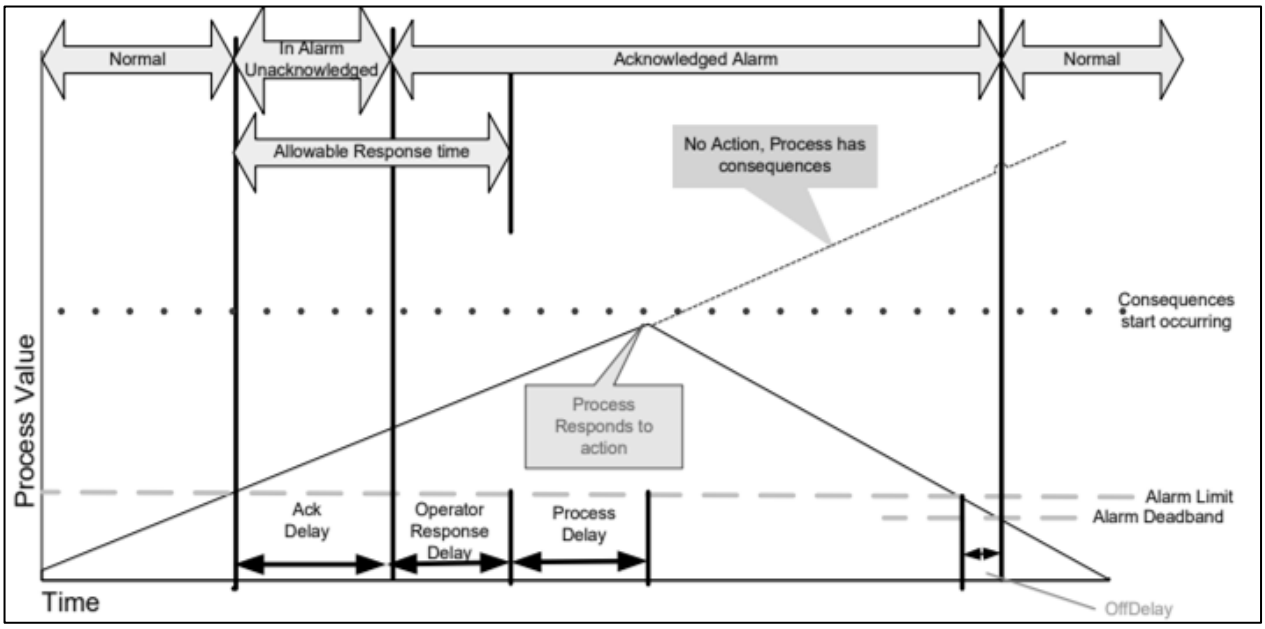


Рисунок 17 – Жизненный цикл тревоги

Реализация

Информация о тревогах и условиях их возникновения (лимиты значений) хранится в БД Genius CM. Когда OPC UA сервер начинает свою работу, менеджер узлов вызывает метод CreateAlarmsObjects, который создает AlarmState для каждого лимита и привязывает его к соответствующей станции аналитики (рисунок 18). По умолчанию, все тревоги создаются в неактивном состоянии.

```
List<AlarmState> CreateAlarmsObjects(
List<AnalysisUnitState> analysisStates, List<AlarmLimit> alarmLimits) {
  var alarmStates = new List<AlarmState>();
  foreach (var alarmLimit in alarmLimits){
    var source = analysisStates.FirstOrDefault(
      a => a.Unit.AnalysisTriple.ID == alarmLimit.Info.AnalysisId &&
           a.Unit.AnalysisTriple.Class == alarmLimit.Info.AnalysisClass);

    if (source != null) {
      var alarmState = new AlarmState(this, source, alarmLimit);
      AddPredefinedNode(SystemContext, alarmState);
      alarmStates.Add(alarmState);
    }
  }
  return alarmStates;
}
```

Рисунок 18 – Создание списка AlarmState

Информация об активных в данный момент тревогах также хранится в БД. За её обновление отвечает класс AlarmProvider. Genius CM не предоставляет API для её получения и никак не информирует сервер о появлении новых записей. AlarmProvider проверяет файл БД каждые несколько секунд на наличие изменений: если обновилось время модификации файла, значит необходимо извлечь новую информацию. Если появились новые записи, генерируется событие OnAlarmsUpdated. Рассмотрим обработку тревог более подробно (рисунок 19).

```
void StartAlarmsMonitoring(
      AlarmProvider alarmProvider,
      List<AlarmState> alarmStates) {

  if (alarmProvider == null)
    return;
  alarmProvider.OnAlarmsUpdated += alarms => {
    lock (Lock) {
      foreach (var alarm in alarms) {
        var relatedState = alarmStates.FirstOrDefault(
                  state => Equals(state.AlarmLimit.Info, alarm.Info));
        if (relatedState != null) {
          relatedState.UpdateAlarm(alarm);
        }
      }
    }
  };
  alarmProvider.StartMonitoring();
}
```

Рисунок 19 – Обработка события OnAlarmsUpdated

Метод StartAlarmsMonitoring вызывается после создания AlarmState. Он инициализирует наблюдение и подписывается на событие OnAlarmsUpdated. Для каждой тревоги выполняется поиск ранее созданного AlarmState. Затем состояние обновляется в соответствии с полученными от AlarmProvider'а данными.

Реализация метода UpdateAlarm представлена на рисунке 20. Каждый переход между состояниями должен сопровождаться уникальными идентификатором события (EventId). Метод устанавливает новый EventId, обновляет время возникновения и срочность тревоги. Для предупреждений срочность устанавливается в значение Medium, для тревог – в High. Также метод формирует сообщение с краткой информацией о происшествии, предельном и зафиксированном значениях. После сохранения нового состояния объекта, генерируется новое OPC-событие, содержащее информации об обновлении.

```
void UpdateAlarm(Alarm alarm)
{
  EventId.Value = Guid.NewGuid().ToByteArray();
  ReceiveTime.Value = Time.Value = alarm.EnabledTime;

  SetActiveState(context, alarm.IsActive);
  SetSeverity(context, alarm.Severity);
  Retain.Value = alarm.IsActive;

  Message.Value = alarm.IsActive
    ? $"{(alarm.Severity > EventSeverity.Medium ? "Alarm" : "Warning")}." +
      $"Extreme value is {alarm.ExtremeValue}
        (limit is {alarm.LimitValue})"
    : null;

  ClearChangeMasks(context, true);

  if (AreEventsMonitored)
  {
    var es = new InstanceStateSnapshot();
    es.Initialize(context, this);
    ReportEvent(context, es);
  }
}
```

Рисунок 20 – Реализация метода UpdateAlarm

3.6 Методы управления станциями анализа

OPC UA добавило возможность объектам экспортировать методы. В терминах OPC, метод – это легковесная функция, область действия которой ограничена объектом, аналогично методу класса в ООП.

Методы вызываются клиентом, исполняются на сервере и возвращают результат обратно клиенту. Жизненный цикл экземпляра метода начинается, когда клиент вызывает метод, и заканчивается, когда возвращается результат. Хотя методы могут влиять на состояние объекта-владельца, они не имеют своего собственного явного состояния. Методы могут иметь различное количество входных аргументов и возвращать результирующие аргументы.

Каждый метод определяется узлом типа MethodNodeClass. Этот узел содержит метаданные, которые описывают аргументы метода и его поведение. Методы вызываются с помощью службы Call Service. Клиенты обнаруживают методы, поддерживаемые сервером, просматривая ссылки на объекты, в которых они указаны.

Реализация

Каждый объект, описывающий станцию аналитики, содержит 3 метода:

- *GetAnalysisStatus* – используется для запроса текущего состояния;
- *StartAnalysis* – запускает станцию аналитики;
- *StopAnalysis* – отключает станцию.

Все три метода используют ApiProvider для взаимодействия с Genius CM Web API. Методы не имеют входных аргументов и возвращают логическое значение, которое определяет результат вызова. Также они содержат название и краткое описание.

Когда клиент вызывает OPC-метод, вызывается функция OnCall (рисунок 21). Она получает системный контекст, ссылку на исходный объект MethodState, а также списки входных и выходных аргументов.

```
ServiceResult OnCall(
      ISystemContext context,
      MethodState method,
      IList<object> inputArguments,
      IList<object> outputArguments)
```

Рисунок 21 – Сигнатура метода OnCall

Система Genius CM предоставляет RESTful API для управления станциями. Необходимо выполнить GET-запрос, в котором указать желаемое действие, класс и идентификатор станции. Результат выполнения будет возвращен в JSON формате. Пример запроса текущего статуса станции и ответа Genius CM представлен на рисунке 22. Другие методы имеют схожую реализацию.

```
[GET] URL: WebApiHost/get_status/<analyticUnitClass>/<analyticUnitId>
Response:
{
  "unit": "F1_AnalyticUnit"
  "status": "running"
}
```

Рисунок 22 – Пример запроса статуса станции

Выводы по разделу

В этой главе были описаны основные сущности OPC UA приложения. Было рассмотрено создание адресного пространства и менеджера узлов, описана теория, лежащая в основе OPC-объектов, их базовые классы и возможности. Также была подробно рассмотрена реализация основных OPC-сервисов.

Были достигнуты следующие результаты:

1) Реализован OPC DA сервис. Сервис был интегрирован с Genius CM. Он получает актуальные технологические и аналитические данные от системы посредством MQTT сообщений.

2) Разработан OPC HDA сервис. Сервис запрашивает информацию из исторической базы данных Genius CM и может предоставить историю отслеживаемых значений для конкретного параметра.

3) Реализован OPC AE сервис. Сервис извлекает информацию из Genius CM Alarms базы данных и может информировать клиентов об тревогах и нештатных ситуациях на производстве.

4) Реализованы методы управления станциями аналитики с помощью OPC UA Method Call. Методы интегрированы с Genius CM Web API.

4 ТЕСТИРОВАНИЕ

Тестирование приложения включало в себя автоматическое модульное и ручное интеграционное тестирование.

Модульное тестирование проводилось с использованием NUnit 3 Framework. NUnit – это фреймворк с открытым исходным кодом, предназначенный для тестирование .Net Framework, .Net Standard и .Net Core приложений. Первая версия фреймворка являлась портированной версией JUnit – широко используемого для тестирования Java-приложений фреймворка. К текущей версии, NUnit 3, библиотека была полностью переписана и получила несколько отличительных особенностей:

- Сильная поддержка тестов на основе данных (DDD testing);
- Поддержка запуска нескольких тестов параллельно;
- Каждый тестовый пример может быть добавлен к одной или нескольким категориям, поддержка выборочного запуска.
- Тесты могут быть запущены из консоли, в Visual Studio, через адаптер или через сторонние программы;

Модульное тестирование охватывает случаи загрузки и обработки данных, такие как анализ файла конфигурации и создание экземпляра класса на его основе. Описание некоторых тестовых случаев приведено в приложении В.

Интеграционное тестирование проводилось с использованием приложения UaExpert Client. Приложение является полнофункциональным OPC UA клиентом, демонстрирующим возможности OPC UA Client SDK. Оно разработано как тестовый клиент общего назначения, поддерживающий функции OPC UA, такие как DataAccess, Alarms & Condition, Historical Access и вызов UA методов. Клиент предоставляет графический интерфейс для отображения данных, поддерживает построение графиков и отчетов.

Клиент позволяет запрашивать данные с сервера, а затем оценивать их визуально. Полученные данные также можно экспортировать в формат CSV.

Примеры некоторых запросов приведены ниже. Запрос исторических данных для трех тегов за 13 марта, с 11:00 до 11:10 показан на рисунке 23.

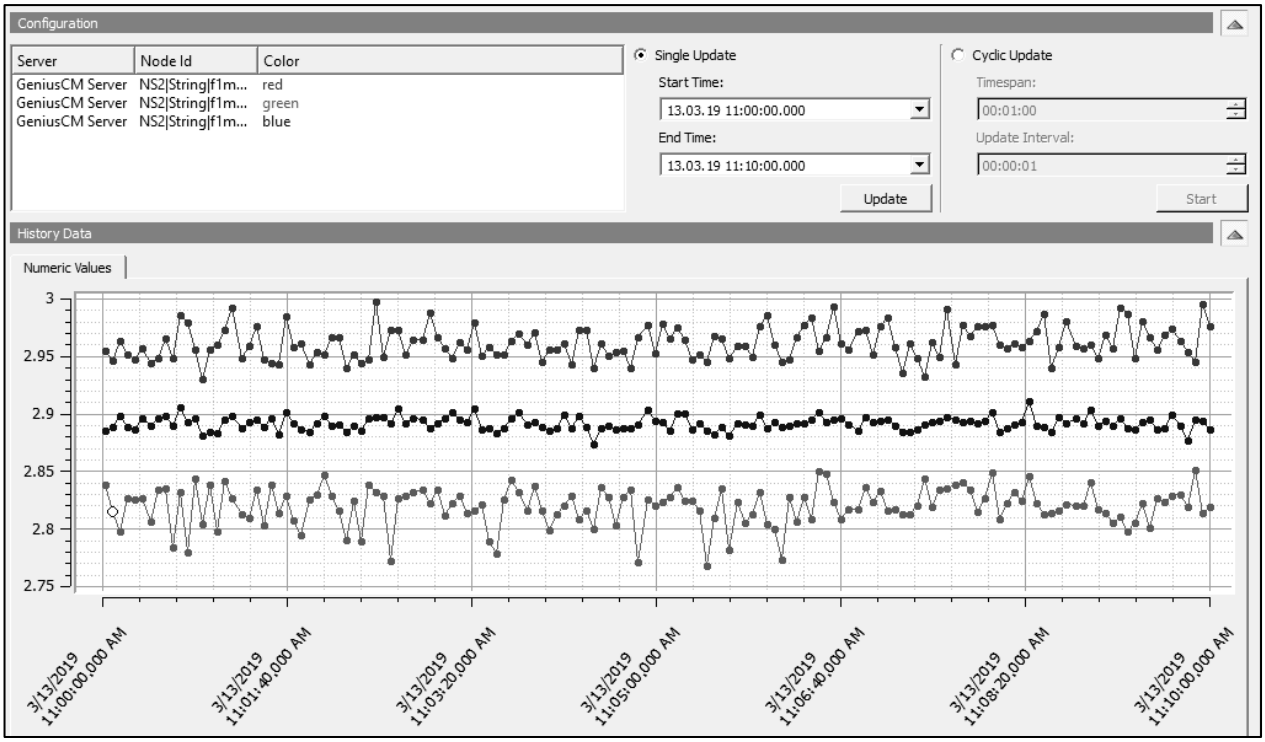


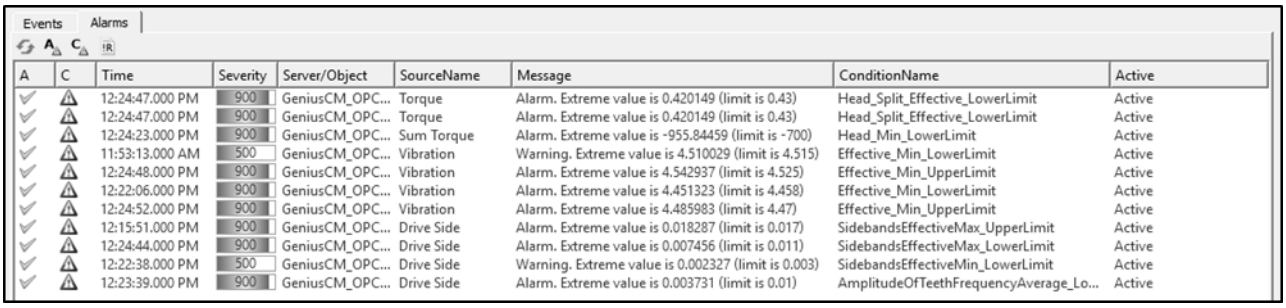
Рисунок 23 – Запрос исторических данных

Результаты работы DA сервиса показаны на рисунке 24. Сервис предоставляет название, значение, тип, метку времени и статус код для каждого наблюдаемого параметра. Если информация не была предоставлена Genius CM, отображается код ошибки. Данные обновляются, как только приложение получает MQTT сообщение от системы.

#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	GeniusCM_OPC...	NS2|String|f1m...	AmplitudeOfTe...	0.006046	Double	11:18:35.482 AM	11:19:03.401 AM	Good
2	GeniusCM_OPC...	NS2|String|f1m...	AmplitudeOfTe...	6	Double	11:18:37.975 AM	11:18:42.942 AM	Good
3	GeniusCM_OPC...	NS2|String|f1m...	AmplitudeOfTe...	0.009353	Double	11:18:37.975 AM	11:19:03.401 AM	Good
4	GeniusCM_OPC...	NS2|String|f1m...	AmplitudeOfTe...	0.002886	Double	11:18:37.975 AM	11:19:03.401 AM	Good
5	GeniusCM_OPC...	NS2|String|f1m...	RotSpeed_Avg	10	Double	11:18:37.975 AM	11:18:38.779 AM	Good
6	GeniusCM_OPC...	NS2|String|f1m...	RotSpeed_Max	10	Double	11:18:37.975 AM	11:18:38.779 AM	Good
7	GeniusCM_OPC...	NS2|String|f1m...	RotSpeed_Min	10	Double	11:18:37.975 AM	11:18:38.779 AM	Good
8	GeniusCM_OPC...	NS2|String|f1m...	SidebandsEffect...	0.016803	Double	11:18:37.975 AM	11:19:03.401 AM	Good
9	GeniusCM_OPC...	NS2|String|f1m...	SidebandsEffect...	6	Double	11:18:37.975 AM	11:18:42.942 AM	Good
10	GeniusCM_OPC...	NS2|String|f1m...	SidebandsEffect...	0.025211	Double	11:18:37.975 AM	11:19:03.401 AM	Good
11	GeniusCM_OPC...	NS2|String|f1m...	SidebandsEffect...	0.010648	Double	11:18:37.975 AM	11:19:03.401 AM	Good
12	GeniusCM_OPC...	NS2|String|f1m...	SidebandsEffect...	0.008633	Double	11:18:37.975 AM	11:19:03.401 AM	Good
13	GeniusCM_OPC...	NS2|String|f1m...	SidebandsEffect...	6	Double	11:18:37.975 AM	11:18:42.942 AM	Good
14	GeniusCM_OPC...	NS2|String|f1m...	SidebandsEffect...	0.010573	Double	11:18:37.975 AM	11:19:03.401 AM	Good
15	GeniusCM_OPC...	NS2|String|f1m...	SidebandsEffect...	0.00595	Double	11:18:37.975 AM	11:19:03.401 AM	Good
16	GeniusCM_OPC...	NS2|String|f1m...	SumHarmonic...	0.013986	Double	11:18:37.975 AM	11:19:03.401 AM	Good
17	GeniusCM_OPC...	NS2|String|f1m...	SumHarmonic...	6	Double	11:18:37.975 AM	11:18:42.942 AM	Good
18	GeniusCM_OPC...	NS2|String|f1m...	SumHarmonic...	0.01998	Double	11:18:37.975 AM	11:19:03.401 AM	Good
19	GeniusCM_OPC...	NS2|String|f1m...	SumHarmonic...	0.009662	Double	11:18:37.975 AM	11:19:03.401 AM	Good
20	GeniusCM_OPC...	NS2|String|f1m...	SumSidebands...		Null	11:18:38.779 AM	11:18:38.779 AM	BadNoDataAvailable
21	GeniusCM_OPC...	NS2|String|f1m...	SumSidebands...		Null	11:18:38.779 AM	11:18:38.779 AM	BadNoDataAvailable
22	GeniusCM_OPC...	NS2|String|f1m...	SumSidebands...	0.054196	Double	11:18:37.975 AM	11:19:03.401 AM	Good
23	GeniusCM_OPC...	NS2|String|f1m...	SumSidebands...	6	Double	11:18:37.975 AM	11:18:42.942 AM	Good
24	GeniusCM_OPC...	NS2|String|f1m...	SumSidebands...	0.077122	Double	11:18:37.975 AM	11:19:03.401 AM	Good
25	GeniusCM_OPC...	NS2|String|f1m...	SumSidebands...	0.035993	Double	11:18:37.975 AM	11:19:03.401 AM	Good
26	GeniusCM_OPC...	NS2|String|f1m...	SumSidebands...	0.03266	Double	11:18:37.975 AM	11:19:03.401 AM	Good
27	GeniusCM_OPC...	NS2|String|f1m...	SumSidebands...	6	Double	11:18:37.706 AM	11:18:42.942 AM	Good

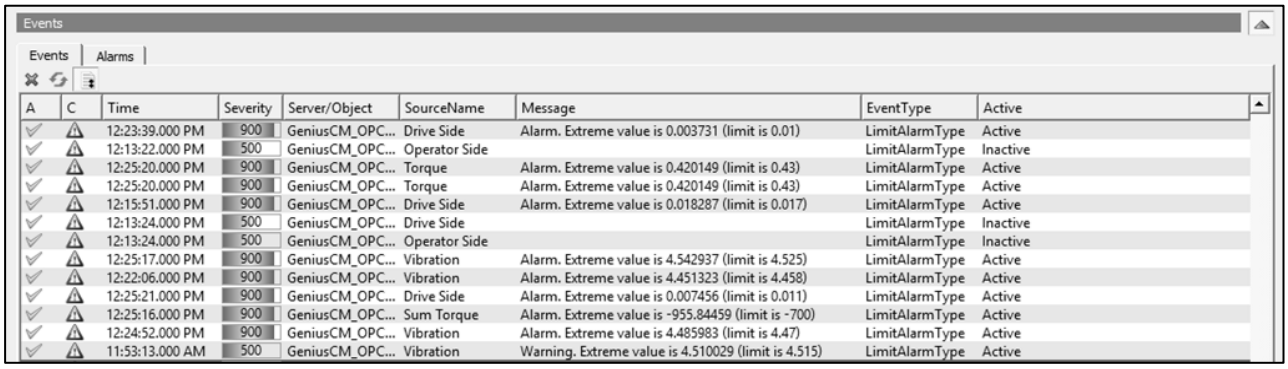
Рисунок 24 – Data Access View

AE сервис предоставляет уведомления о тревогах, активных в данный момент (рисунок 25), и возникающих событиях (рисунок 26). Для каждой тревоги отображается время её возникновения, срочность и сообщение с краткой информацией. Данные обновляются каждые несколько секунд. События отражают переходы между состояниями тревоги. Например, при вызове метода Acknowledge, генерируется соответствующее событие.



A	C	Time	Severity	Server/Object	SourceName	Message	ConditionName	Active
		12:24:47.000 PM	900	GeniusCM_OPC...	Torque	Alarm. Extreme value is 0.420149 (limit is 0.43)	Head_Split_Effective_LowerLimit	Active
		12:24:47.000 PM	900	GeniusCM_OPC...	Torque	Alarm. Extreme value is 0.420149 (limit is 0.43)	Head_Split_Effective_LowerLimit	Active
		12:24:23.000 PM	900	GeniusCM_OPC...	Sum Torque	Alarm. Extreme value is -955.84459 (limit is -700)	Head_Min_LowerLimit	Active
		11:53:13.000 AM	500	GeniusCM_OPC...	Vibration	Warning. Extreme value is 4.510029 (limit is 4.515)	Effective_Min_LowerLimit	Active
		12:24:48.000 PM	900	GeniusCM_OPC...	Vibration	Alarm. Extreme value is 4.542937 (limit is 4.525)	Effective_Min_UpperLimit	Active
		12:22:06.000 PM	900	GeniusCM_OPC...	Vibration	Alarm. Extreme value is 4.451323 (limit is 4.458)	Effective_Min_LowerLimit	Active
		12:24:52.000 PM	900	GeniusCM_OPC...	Vibration	Alarm. Extreme value is 4.485983 (limit is 4.47)	Effective_Min_UpperLimit	Active
		12:15:51.000 PM	900	GeniusCM_OPC...	Drive Side	Alarm. Extreme value is 0.018287 (limit is 0.017)	SidebandsEffectiveMax_UpperLimit	Active
		12:24:44.000 PM	900	GeniusCM_OPC...	Drive Side	Alarm. Extreme value is 0.007456 (limit is 0.011)	SidebandsEffectiveMax_LowerLimit	Active
		12:22:38.000 PM	500	GeniusCM_OPC...	Drive Side	Warning. Extreme value is 0.002327 (limit is 0.003)	SidebandsEffectiveMin_LowerLimit	Active
		12:23:39.000 PM	900	GeniusCM_OPC...	Drive Side	Alarm. Extreme value is 0.003731 (limit is 0.01)	AmplitudeOfTeethFrequencyAverage_Lo...	Active

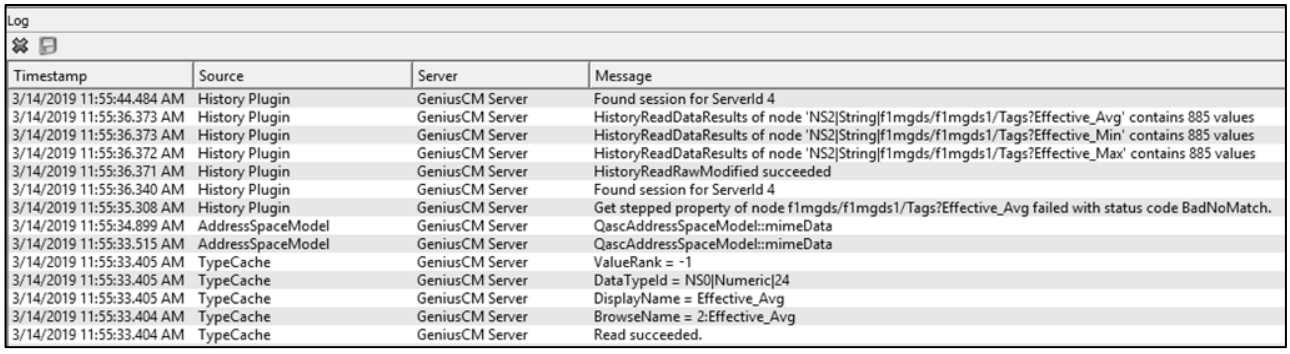
Рисунок 25 – Просмотр тревог



A	C	Time	Severity	Server/Object	SourceName	Message	EventType	Active
		12:23:39.000 PM	900	GeniusCM_OPC...	Drive Side	Alarm. Extreme value is 0.003731 (limit is 0.01)	LimitAlarmType	Active
		12:13:22.000 PM	500	GeniusCM_OPC...	Operator Side		LimitAlarmType	Inactive
		12:25:20.000 PM	900	GeniusCM_OPC...	Torque	Alarm. Extreme value is 0.420149 (limit is 0.43)	LimitAlarmType	Active
		12:25:20.000 PM	900	GeniusCM_OPC...	Torque	Alarm. Extreme value is 0.420149 (limit is 0.43)	LimitAlarmType	Active
		12:15:51.000 PM	900	GeniusCM_OPC...	Drive Side	Alarm. Extreme value is 0.018287 (limit is 0.017)	LimitAlarmType	Active
		12:13:24.000 PM	500	GeniusCM_OPC...	Drive Side		LimitAlarmType	Inactive
		12:13:24.000 PM	500	GeniusCM_OPC...	Operator Side		LimitAlarmType	Inactive
		12:25:17.000 PM	900	GeniusCM_OPC...	Vibration	Alarm. Extreme value is 4.542937 (limit is 4.525)	LimitAlarmType	Active
		12:22:06.000 PM	900	GeniusCM_OPC...	Vibration	Alarm. Extreme value is 4.451323 (limit is 4.458)	LimitAlarmType	Active
		12:25:21.000 PM	900	GeniusCM_OPC...	Drive Side	Alarm. Extreme value is 0.007456 (limit is 0.011)	LimitAlarmType	Active
		12:25:16.000 PM	900	GeniusCM_OPC...	Sum Torque	Alarm. Extreme value is -955.84459 (limit is -700)	LimitAlarmType	Active
		12:24:52.000 PM	900	GeniusCM_OPC...	Vibration	Alarm. Extreme value is 4.485983 (limit is 4.47)	LimitAlarmType	Active
		11:53:13.000 AM	500	GeniusCM_OPC...	Vibration	Warning. Extreme value is 4.510029 (limit is 4.515)	LimitAlarmType	Active

Рисунок 26 – Просмотр событий

В дополнение, UaExpert Client ведет логирование своих действий и действий сервера (рисунок 27). Это позволяет отслеживать вызов методов и возвращаемые ими значения.



Timestamp	Source	Server	Message
3/14/2019 11:55:44.484 AM	History Plugin	GeniusCM Server	Found session for ServerId 4
3/14/2019 11:55:36.373 AM	History Plugin	GeniusCM Server	HistoryReadDataResults of node 'NS2|String|f1mgds/f1mgds1/Tags?Effective_Avg' contains 885 values
3/14/2019 11:55:36.373 AM	History Plugin	GeniusCM Server	HistoryReadDataResults of node 'NS2|String|f1mgds/f1mgds1/Tags?Effective_Min' contains 885 values
3/14/2019 11:55:36.372 AM	History Plugin	GeniusCM Server	HistoryReadDataResults of node 'NS2|String|f1mgds/f1mgds1/Tags?Effective_Max' contains 885 values
3/14/2019 11:55:36.371 AM	History Plugin	GeniusCM Server	HistoryReadRawModified succeeded
3/14/2019 11:55:36.340 AM	History Plugin	GeniusCM Server	Found session for ServerId 4
3/14/2019 11:55:35.308 AM	History Plugin	GeniusCM Server	Get stepped property of node f1mgds/f1mgds1/Tags?Effective_Avg failed with status code BadNoMatch.
3/14/2019 11:55:34.899 AM	AddressSpaceModel	GeniusCM Server	QascAddressSpaceModel::mimeData
3/14/2019 11:55:33.515 AM	AddressSpaceModel	GeniusCM Server	QascAddressSpaceModel::mimeData
3/14/2019 11:55:33.405 AM	TypeCache	GeniusCM Server	ValueRank = -1
3/14/2019 11:55:33.405 AM	TypeCache	GeniusCM Server	DataTypeId = NS0|Numeric|24
3/14/2019 11:55:33.405 AM	TypeCache	GeniusCM Server	DisplayName = Effective_Avg
3/14/2019 11:55:33.404 AM	TypeCache	GeniusCM Server	BrowseName = 2:Effective_Avg
3/14/2019 11:55:33.404 AM	TypeCache	GeniusCM Server	Read succeeded.

Рисунок 27 – Просмотр логов

ЗАКЛЮЧЕНИЕ

В работе продемонстрировано проектирование и реализация OPC UA приложения-сервера. Приложение было успешно интегрировано с системой мониторинга состояний Genius CM.

Основные результаты

1. Проведен анализ предметной области и обзор аналогов. Описаны протоколы OPC Classis и OPC UA, рассмотрены системы мониторинга и сбора технологических данных.

2. Разработана архитектура системы. Рассмотрены функциональные и нефункциональные требования, варианты ее использования отражены на use-case диаграмме. Создана доменная модель приложения, отражающая основные компоненты приложения и взаимосвязи между ними.

3. Было реализовано OPC UA приложение, предоставляющее DA, HDA и AE сервисы. Сервисы обеспечивают доступ к технологическим и аналитическим данным, поддерживают запросы исторических данных, предоставляют уведомления о тревогах и нештатных ситуациях.

4. Приложение было протестировано: проведено модульное и интеграционное тестирование. OPC UA сервер прошел все тесты и готов к использованию.

Направления дальнейших исследований

Дальнейшее развитие может быть направлено на расширение функциональности сервера и более полную его интеграцию с системой Genius CM:

- Улучшение системы авторизации. Возможно разделение прав пользователей и введение ролей. Кроме того, может быть реализована авторизация на основе сертификатов.
- Доработка взаимодействия с подсистемами Genius CM: получение информации о тревоге может быть реализовано через MQTT, аналогично DA-сервису. Однако это потребует модификаций самой Genius CM.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. B. Albahari, J. Albahari C# 7.0 in a Nutshell // O'Reilly Media, 2017 – P. 214-220.

2. Genius Condition Monitoring System. [Electronic resource] URL: https://www.sms-group.com/technical-service/service-modules/genius-cm (date of access: 15.02.2019).

3. GPLv2 License Text. [Electronic resource] URL: https://opcfoundation.org/license/gpl.html (date of access: 13.01.2019).

4. IBA Analyzer. [Electronic resource] URL: https://www.iba-ag.com/ru/products/ibaanalyzer (date of access: 21.01.2019).

5. IBA Monitoring System. [Electronic resource] URL: https://www.iba-ag.com/ru/iba-system (date of access: 24.02.2019).

6. IBA PDA Station. [Electronic resource] URL: https://www.iba-ag.com/ru/products/ibapda (date of access: 24.02.2019).

7. IBA System. Acquire & Record. [Electronic resource] URL: https://www.iba-ag.com/en/iba-system/acquire-record (date of access: 24.02.2019).

8. J. Berge Software for Automation: Architecture, Integration, and Security // ISA – The Instrumentation, Systems and Automation Society, 2005 – P. 97 – 140.

9. J. Mesnil Mobile and Web Messaging // O'Reilly Media Inc., 2014 – P. 29-34.

10. OPC Classis Standard. [Electronic resource] URL: https://opcfoundation.org/faq/what-is-opc-classic (date of access: 16.02.2019).

11. OPC Foundation OPC Unified Architecture Part 1: Overview and Concepts // Industry Standard Specification, 2017 – 30 p.

12. OPC Foundation OPC Unified Architecture Part 11: Historical Data Access // Industry Standard Specification, 2017 – 49 p.

13. OPC Foundation OPC Unified Architecture Part 3: Address Space Model // Industry Standard Specification, 2017 – 112 p.

14. OPC Foundation OPC Unified Architecture Part 4: Services // Industry Standard Specification, 2017 – 201 p.

15. OPC Foundation OPC Unified Architecture Part 5: Information Model // Industry Standard Specification, 2017 – 158 p.

16. OPC Foundation OPC Unified Architecture Part 8: Data Access // Industry Standard Specification, 2017 – 48 p.

17. OPC Foundation OPC Unified Architecture Part 9: Alarms and Conditions // Industry Standard Specification, 2017 – 125 p.

18. OPC Unified Architecture .NET Standard. [Electronic resource] URL: https://github.com/OPCFoundation/UA-.NETStandard (date of access: 19.02.2019).

19. P. Freeman, D. Hart. A Science of design for software-intensive systems // Communications of the ACM. 47, 2004 – P. 19-21.

20. R. Bartos, S. Brockmann, G. Endemann, et al. Steel Manual // VerlagStahleisen GmbH, Düsseldorf, 2015. – 220 p.

21. UA Modeler. [Electronic resource] URL: https://www.unified-automation.com/products/development-tools/uamodeler.html (date of access: 07.03.2019).

22. W. Mahnke, S. Leitner, M. Damm OPC Unified Architecture // Springer Science & Business Media, 2009 – 339 p.

23. What is Industry 4.0? [Electronic resource] URL: https://www.epicor.com/resources/articles/what-is-industry-4-0.aspx (date of access: 25.01.2019).

24. What is OPC? [Electronic resource] URL: https://opcfoundation.org/about/what-is-opc (date of access: 28.01.2019).

25. А. Троелсен, П. Джепикс. Язык программирования C# 7 и платформы .NET и .NET Core. – СПб: Вильямс, 2018. – 149 – 312 стр.

26. Дж. Арлоу, А. Нейштадт. UML 2 и Унифицированный процесс. – М: Символ-Плюс, 2008. – 89 – 98 стр.

27. Официальный сайт OPC Foundation. [Электронный ресурс] URL: https://opcfoundation.org (дата обращения: 27.01.2019).

28. Официальный сайт UA Experts. [Электронный ресурс] URL: https://www.unified-automation.com (дата обращения: 04.03.2019).

29. Э. Эванс. Предметно-Ориентированное проектирование. – СПб: Вильямс, 2011. – 27 – 75 стр.

ПРИЛОЖЕНИЕ А

Фрагмент конфигурации аппаратной системы завода

```
<PlantNode Label="Plant" ID="0">
  <Children>
    <PlantNode Label="RM" ID="rm">
      <Children>
        <PlantNode Label="Top Spindle" ID="rm/tt">
          <Analysis Class="AnalysisTorque" ID="1001" Name="RMTop" />
        </PlantNode>
        <PlantNode Label="Top Sp. Rainflow" ID="rm/trf">
          <Analysis Class="AnalysisRainflow" ID="1001" Name="RMTop" />
        </PlantNode>
        <PlantNode Label="Bottom Spindle" ID="rm/bt">
          <Analysis Class="AnalysisTorque" ID="1002" Name="RMBottom" />
        </PlantNode>
        ...
      </Children>
    </PlantNode>
    <PlantNode Label="F1 Main Gearbox" ID="f1/mg">
      <Children>
        <PlantNode Label="Drive Side" ID="f1mgds">
          <Children>
            <PlantNode Label="Input Shaft" ID="f1mgds1">
              <Analysis Class="AnalysisBearing" ID="100201"
                        Name="f1MainGearboxDriveSideInputShaft" />
            </PlantNode>
            <PlantNode Label="Output Shaft" ID="f1mgds3">
              <Analysis Class="AnalysisBearing" ID="100202"
                        Name="f1MainGearboxDriveSideOutputShaft" />
            </PlantNode>
            <PlantNode Label="Vibration" ID="f1mgdsv">
              <Analysis Class="AnalysisVibration" ID="1002030000"
                        Name="f1MainGearboxDriveSide" />
            </PlantNode>
          </Children>
        </PlantNode>
      </Children>
    </PlantNode>
    <PlantNode Label="F1 Pinion Gearbox" ID="f1/pg"> ... </PlantNode>
    <PlantNode Label="F2 Main Gearbox" ID="f2/mg"> ... </PlantNode>
  </Children>
</PlantNode>
```

ПРИЛОЖЕНИЕ Б

Фрагмент конфигурации станции анализа

```
<analysisSchema>
  <analysis class="AnalysisAcousticBearing">
    <column name="Effective_Avg" group="Effective" show="g" unit="V" />
    <column name="Effective_Min" group="Effective" show="g" unit="V" />
    <column name="InnerRingAmp_Avg" group="InnerRing" show="g" unit="V" />
    <column name="InnerRing_Avg" group="InnerRing" show="d" unit="1" />
      ...
  </analysis>
  <analysis class="AnalysisAqua">
    <column name="Average" group="Aqua" show="g" unit="%" />
    <column name="Count" group="Aqua" show="t" unit="1" />
    <column name="Max" group="Aqua" show="g" unit="%" />
    <column name="Min" group="Aqua" show="g" unit="%" />
      ...
  </analysis>
  <analysis class="AnalysisAxialPistonPumpPressure">...</analysis>
  <analysis class="AnalysisAxialPistonPumpVibration">...</analysis>
  <analysis class="AnalysisBearing">...</analysis>
  <analysis class="AnalysisBearingRev">...</analysis>
  ...
</analysisSchema>
```

ПРИЛОЖЕНИЕ В

Протокол тестирования приложения

Таблица 1 – Протокол тестирования

№	Тип данных	Значение
1	Входные данные	Загрузка структуры завода из XML файла. Структура может включать описание нескольких станков и станций аналитики. Устройства могут быть объединены в сегменты.
	Ожидаемый результат	Экземпляр класса UnderlyingSystem должен быть создан. Экземпляр должен содержать корректную структуру завода. Корневой блок должен содержать все сегменты первого уровня. Параметры устройств и станций анализа должны соответствовать схеме.
	Результат теста	Пройден.
2	Входные данные	Разбор сообщения об обновлении данных от системы Genius CM. Сообщение содержит класс и словарь данных. Значения для некоторых тегов могут отсутствовать.
	Ожидаемый результат	Сообщение должно быть разобрано корректно. Новые значения из словаря данных должны быть присвоены соответствующим тегам.
	Результат теста	Пройден.
3	Входные данные	Запрос истории значений тега за определенный промежуток времени. Значения за этот период могут отсутствовать в БД.
	Ожидаемый результат	Данные должны быть прочитаны верно. Должен быть создан экземпляр класса HistoryData, содержащий список DataValue.
	Результат теста	Пройден.

Окончание таблицы 1

№	Тип данных	Значение
4	Входные данные	Запрос онлайн-данных определенного тега. Данные могут быть недоступны.
	Ожидаемый результат	Если данные недоступны, должен быть возвращен код BadNoData. Иначе экземпляр MonitoringItem должен быть создан. Значение тега должно обновляться при получении нового сообщения от Genius CM.
	Результат теста	Пройден.
5	Входные данные	Вызов OPC UA метода для запроса текущего статуса станции наблюдения.
	Ожидаемый результат	Должен быть произведен вызов Genius CM Web API. Результат должен быть разобран, код станции возвращен.
	Результат теста	Пройден.
6	Входные данные	Подтверждение сообщения о тревоге.
	Ожидаемый результат	Тревога должна быть закрыта (переведена в неактивное положение). Должна быть создана соответствующая запись в БД и OPC событие.
	Результат теста	Пройден.
7	Входные данные	Авторизация пользователя. Пользователь указывает логин и пароль и запрашивает подключение к серверу.
	Ожидаемый результат	Если логин отсутствует в БД или хеш пароля не совпадает с имеющимся, должен быть возвращен код BadUnauthorized. Иначе пользователь должен получить доступ к серверу.
	Результат теста	Пройден.