Библиотека поддержки протокола OPC UA для программируемых логических контроллеров семейства «Багет»

Т. К. Грингауз[1], Д. В. Яриков[2]

[1]НИЦ «Курчатовский институт» – НИИСИ, Москва, Россия, gring@niisi.ras.ru;

[2]НИЦ «Курчатовский институт» – НИИСИ, Москва, Россия, yarikov@ niisi.ras.ru

Аннотация. В среду отечественной операционной системы реального времени семейства Багет портирована свободно распространяемая библиотека open62541, реализующая протокол OPC UA. В статье приведен обзор базовых понятий OPC UA, перечислены функциональные возможности портированной версии библиотеки, описана модельная задача, приведен разбор ключевых фрагментов ее кода.

Ключевые слова: программируемый логический контроллер, протокол OPC UA, ОСРВ Багет

1. Введение

В ФГУ ФНЦ НИИСИ РАН ведется разработка линейки программируемых логических контроллеров на отечественной электронной компонентной базе (далее – «контроллеры семейства Багет»). Контроллеры предназначены для использования в автоматизированных системах управления технологическими процессами. В состав программного обеспечения контроллеров входит программа «Библиотека поддержки протокола OPC UA» (БПП OPC UA). Библиотека может использоваться приложениями, функционирующими в среде операционной системы реального времени ОСРВ Багет 2.7 [1], для предоставления или потребления данных посредством OPC UA[2].

Стандарт OPC UA (Unified Archtecture), известный как IEC 62541, определяет средства информационного моделирования данных и коммуникационный протокол для промышленных систем. Стандарт поддерживается некоммерческим Консорциумом OPC Foundation. Первая спецификация стандарта была представлена в 2006 г., а в 2012 г. стандарт был оформлен в рамках Единого реестра европейских стандартов [3]. К сильным сторонам стандарта, способствующим его широкому применению, относятся: аппаратно- и платформонезависимость, возможность построения на его основе сложных информационных моделей, сервис-ориентированная архитектура. Сервисы, специфицированные в OPC UA, определяют информацию, которой должны обмениваться приложения, но не конкретный способ коммуникации по сети, а также не конкретную реализацию посредством API.

Библиотека БПП OPC UA разработана на базе открытого исходного кода свободно распространяемого продукта open62541 [4]. Библиотека open62541 реализует стек бинарного протокола OPC UA, а также набор функций (SDK) для разработки на языке Си клиентских и серверных приложений OPC UA.

В статье приведен обзор базовых понятий OPC UA, перечислены функциональные возможности БПП OPC UA, описана модельная задача, приведен разбор ключевых фрагментов ее кода.

Изложение материала в разделе 2 основано на источниках [2],[5],[6].

2. Обзор OPC UA

2.1. Компоненты унифицированной архитектуры OPC UA

Унифицированная архитектура OPC UA включает в себя:

- метамодель для информационного моделирования, которая сочетает объектно-ориентированность с возможностью установления семантических отношений между объектами;
- асинхронный протокол (построенный на TCP, HTTP или SOAP), который определяет обмен сообщениями в рамках сессий, установленных над защищенными каналами связи;
- систему типов данных и схемы (двоичную и основанную на XML) их кодирования в сообщениях;
- набор из 37 стандартных сервисов для взаимодействия с серверными информационными моделями.

Совокупность механизмов кодирования (форматирования), защищенного канала и транспорта принято называть «стеком OPC UA».

Сервисы OPC UA - это интерфейс между серверами (поставщиками информационной модели) и клиентами (потребителями этой информационной модели). Сервисы используют стек OPC UA для обмена данными между клиентом и сервером.

2.2. Информационное моделирование в OPC UA

2.2.1. Информационная модель. Объектная модель. Адресное пространство

Информационная модель определяет, характеризует и связывает информационные ресурсы системы или набора систем, информацию о которых сервер предоставляет клиентам. Стандарт OPC UA специфицирует структуру объектно-ориентированных информационных моделей, предоставляемых сервером, и протокол, с помощью которого клиент может взаимодействовать с информационной моделью по сети.

В информационной модели OPC UA объект определяется как набор переменных, методов и ссылок на другие объекты. Объекты OPC UA типизированы. Тип определяет не только структуру объекта, но и его семантику. Для доступа к объектам и их компонентам клиент вызывает сервисы OPC UA. Сервисы обеспечивают чтение и запись переменных, вызов методов, создание экземпляров и удаление объектов, подписку на уведомления об изменениях и другие действия с объектами.

В OPC UA определен стандартный способ представления информационных моделей. Введено понятие «адресное пространство сервера» (далее – «адресное пространство»). Адресное пространство определено как «вся информация (collection of information), которую сервер делает видимой для своих клиентов» [2]. Информационная модель в адресном пространстве OPC UA представляется как граф, состоящий из узлов и ссылок между ними. Объект и его компоненты образуют связный подграф.

2.2.2. Модель узла

Узел определяется как набор атрибутов и ссылок. Атрибуты описывают узел, а ссылки определяют его связи (отношения) с другими узлами. В состав атрибутов входит обязательный атрибут NodeId - уникальный идентификатор узла внутри сервера.

2.2.3. Классы узлов

Узлы классифицированы. Определены 8 классов узлов:

-Variable (переменная);
-VariableType (тип переменной);
-Object (объект);
-ObjectType (тип объекта);
-ReferenceType (тип ссылки);
-DataType (тип данных);
-Method (метод);
-View (представление).

Каждый узел в адресном пространстве является экземпляром одного из восьми классов. Класс узла определяет состав набора его атрибутов и ссылок. Разные экземпляры одного класса имеют одинаковый набор атрибутов, различаться могут только их значения. Клиентам и серверам не разрешается определять дополнительные классы узлов или расширять список атрибутов для класса узла.

Класс узла идентифицируется атрибутом NodeClass:

```
typedef enum {
UA_NODECLASS_UNSPECIFIED = 0,
UA_NODECLASS_OBJECT = 1,
UA_NODECLASS_VARIABLE = 2,
UA_NODECLASS_METHOD = 4,
UA_NODECLASS_OBJECTTYPE = 8,
UA_NODECLASS_VARIABLETYPE = 16,
UA_NODECLASS_REFERENCETYPE = 32,
UA_NODECLASS_DATATYPE = 64,
UA_NODECLASS_VIEW = 128,
UA_NODECLASS_FORCE32BIT = 0x7fffffff
} UA_NodeClass;
```

Примечание – Здесь и далее для обозначения стандартных констант, атрибутов, типов данных и др. используются их имена из исходных текстов open62541.

2.2.4. Атрибуты

Атрибуты – это элементарные компоненты классов узлов. Они не видны напрямую в адресном пространстве, но их значения для конкретного узла могут быть запрошены или изменены клиентом посредством сервисов атрибутов (Atribute Service Set).

В спецификациях [2] определение атрибута включает в себя поля: идентификатор атрибута, имя, описание, типа данных и индикатор «обязательного/опциональный».

Каждому атрибуту присвоен числовой идентификатор UA_AttributeId. Всего в OPC UA определено 27 атрибутов с уникальными значениями идентификатора UA_AttributeId. В файле open62541/common.h определен тип данных UA_AttributeId (перечисление). Семантика большинства атрибутов будет описана при рассмотрении классов узлов в разделе 2.2.6 настоящей статьи.

Набор атрибутов, определенный для класса узла, не может расширяться клиентами или серверами. При создании узла в адресном пространстве необходимо указать значения обязательных

атрибутов его класса.

Следующие 7 атрибутов используются в определениях всех классов узлов (первые 4 обязательны, последние 3 – опциональны):

- NodeId (тип данных NodeId) - иденти-фикатор узла;
- NodeClass (тип данных NodeClass) – идентификатор класса узла);
- BrowseName (тип данных QualifiedName) – имя узла для сервера (не локализовано);
- DisplayName (тип данных LocalizedText) – имя узла для клиента (локализовано);
- Description (тип данных LocalizedText) – текстовое описание узла;
- WriteMask (тип данныхUint32) – специфицирует атрибуты, которые доступны для модификации клиентом;
- UserWriteMask (тип данныхUint32) – специфицирует атрибуты, которые доступны для модификации пользователем, подключенным в текущий момент.

Ниже приведены описания наборов атрибутов (из файла open62541/types.h), специфичных для отдельных классов узлов в БПП OPC UA:

```
typedef struct {
UA_UInt32 specifiedAttributes;
UA_LocalizedTextdisplayName;
UA_LocalizedText description;
UA_UInt32 writeMask;
UA_UInt32 userWriteMask;
UA_BooleanisAbstract;
UA_Boolean symmetric;
UA_LocalizedTextinverseName;
} UA_ReferenceTypeAttributes;

typedef struct {
UA_UInt32 specifiedAttributes;
UA_LocalizedTextdisplayName;
UA_LocalizedText description;
 UA_UInt32 writeMask;
UA_UInt32 userWriteMask;
UA_BooleanisAbstract;
} UA_ObjectTypeAttributes;

typedef struct {
UA_UInt32 specifiedAttributes;
UA_LocalizedTextdisplayName;
UA_LocalizedText description;
UA_UInt32 writeMask;
UA_UInt32 userWriteMask;
UA_ByteeventNotifier;
} UA_ObjectAttributes;

typedef struct {
UA_UInt32 specifiedAttributes;
UA_LocalizedTextdisplayName;
UA_LocalizedText description;
UA_UInt32 writeMask;
UA_UInt32 userWriteMask;
UA_Variant value;
UA_NodeIddataType;
UA_Int32 valueRank;
size_tarrayDimensionsSize;
UA_UInt32 *arrayDimensions;
UA_ByteaccessLevel;
UA_ByteuserAccessLevel;
UA_DoubleminimumSamplingInterval;
UA_Booleanhistorizing;
} UA_VariableAttributes;

typedef struct {
UA_UInt32 specifiedAttributes;
UA_LocalizedTextdisplayName;
UA_LocalizedText description;
UA_UInt32 writeMask;
UA_UInt32 userWriteMask;
UA_Variant value;
UA_NodeIddataType;
UA_Int32 valueRank;
size_t arrayDimensionsSize;
UA_UInt32 *arrayDimensions;
UA_BooleanisAbstract;
} UA_VariableTypeAttributes;

typedef struct {
UA_UInt32 specifiedAttributes;
UA_LocalizedTextdisplayName;
UA_LocalizedText description;
UA_UInt32 writeMask;
UA_UInt32 userWriteMask;
UA_Boolean executable;
UA_BooleanuserExecutable;
} UA_MethodAttributes;
```

2.2.5. Ссылки.

Ссылки используются для определения отношений узлов друг с другом. Тип ссылки является экземпляром класса узлов ReferenceType.

Ссылка представляет собой тройку вида <идентификатор узла-источника, идентификатор узла типа (ReferenceType), идентификатор узла-цели>.

Примером ссылки между узлами является ссылка типа «hasTypeDefinition» между узлами классов «переменная» и «тип переменной».

2.2.6. Определения классов узлов

2.2.6.1. Переменные (VariableNode)

Переменные используются для представления содержимого объекта, его реальных данных. Для этого класса специфичны следующие атрибуты:

- value (тип данных UA_Variant) – обязательный. Фактическое значение переменной. Тип данных значения определяется атрибутами DataType, ValueRank и ArrayDimensions;

- dataType (тип данных UA_NodeId) –обязательный. Идентификатор узла класса DataType;

- valueRank (тип данных UA_Int32) –обязательный. Указавает, является ли переменная массивом. Если является, то указывает число размерностей;

- arrayDimensions (тип данных UA_UInt32*) - опциональный. Для массива указывает число элементов по каждой размерности;

- accessLevel (тип данных UA_Byte) –обязательный. Битовая маска, указывающая, доступны ли на чтение и запись текущее значение и его история;

- userAccessLevel (тип данных UA_Byte) – обязательный.Битовая маска, указывающая, доступны ли на чтение и запись текущее значение и его история с учетом прав пользователя;

- minimumSamplingInterval (тип данных UA_Double) - опциональный. Минимальное время, за которое допустимо обновление значения переменной на сервере;

- historizing (тип данных UA_Boolean) – обязательный. Указывает, ведется ли сбор исторических данных в текущий момент.

Определены два вида переменных: свойства (Properties) и переменные данных (DataVariables). Свойства связаны с родительским узлом ссылкой типа "hasProperty" и не могут иметь дочерних узлов. Переменные данных могут иметь в качестве дочерних узлов свойства (тип ссылки "hasProperty") и другие переменные данных (тип ссылки "hasComponent").

2.2.6.2. Типы переменных (VariableTypeNode)

Узлы класса VariableNode создаются как дочерние по отношению к узлам класса VariableTypesNode (ссылка "hasTypeDefinition"). Родительские узлы определяют тип создаваемых переменных и накладывают ограничения на допустимые значения атрибутов dataType;valueRank, arrayDimensions для экземпляров переменных. Кроме того, при создании экземпляра переменной определенного типа дочернему узлу передается семантическая информация от родителя.

2.2.6.3. Объекты (ObjectNode)

Объекты используются для представления систем, компонентов систем, объектов реального мира и программных объектов. Объекты являются экземплярами типа объекта и могут содержать переменные, методы и другие объекты.

2.2.6.4. Типы объектов (ObjectTypeNode)

Класс узла ObjectType используется для представления типа объектов в адресном пространстве сервера. Узлы класса ObjectTypes аналогичны классам в объектно-ориентированных языках. Для класса узла ObjectType специфичен обязательный атрибут isAbstract (тип данных UA_Boolean). Он указывает, является ли тип абстрактным. Абстрактные типы не могут порождать экземпляров объектов.

Пример типа объекта – FolderType (тип «папка»). Объекты этого типа используются для представления адресного пространства в виде иерархии узлов. На рис. 1 изображен встроенный набор папок, задающий предопределенную структуру адресного пространства.

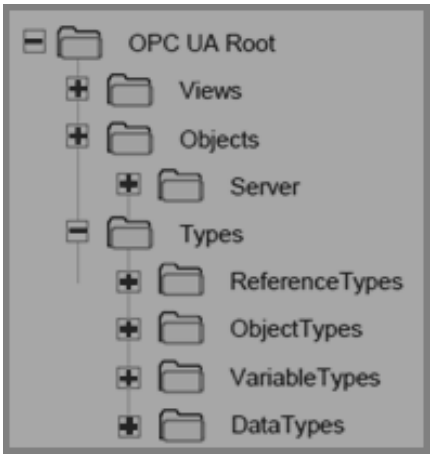

Рис.1 Стандартная структура адресного пространства ([2], Part 5)

2.2.6.5. Типы ссылок (ReferenceTypeNode)

Каждая ссылка между двумя узлами имеет тип, который определяет семантику отношения. Тип ссылки определяется узлом класса ReferenceType. Этот узел является родительским при создании ссылки.

Стандарт OPC UA определяет набор встроенных узлов ReferenceTypes. Их иерархия показана на рисунке 2 (стрелки указывают на отношение «hasSubType»). Полное описание семантики каждого типа ReferenceType содержится в части 3 спецификации OPC UA. Абстрактные типы ReferenceTypes не могут использоваться в реальных ссылках и применяются только для структурирования иерархии. Симметричные ссылки имеют одинаковый смысл для исходного и целевого узлов.

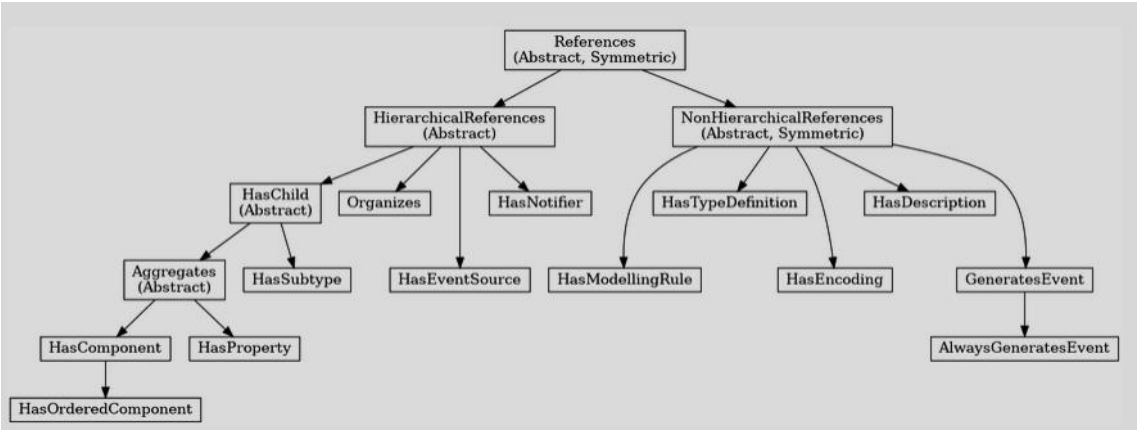

Рис. 2. Иерархия стандартных типов ссылки OPC UA (рисунок заимствован из [4])

2.2.6.6. Типы данных (DataTypeNode)

Узлы класса DataTypes представляют простые и структурированные типы данных. Типы DataTypes могут включать в себя массивы, но они всегда описывают структуру одного экземпляра. Абстрактные типы DataTypes (например, Number) не могут быть типом фактических значений. Они используются для ограничения значений набором дочерних DataTypes (например, UInt32).

2.2.6.7. Методы (MethodNode)

Методы определяют функции, которые вызываются с помощью сервиса Call. Узел MethodNode может иметь специальные свойства для определения входных и выходных аргументов. Эти свойства моделируются как дочерние узлы класса Variable со ссылкой hasProperty. Атрибут QualifiedName для этих переменных принимает значения (0, "InputArguments"), и (0, "OutputArguments") соответственно (стандартные имена, определенные в системном пространстве имен). Более подробно о пространствах имен написано в разделе 2.2.7 настоящей статьи.

Входные и выходные аргументы описываются как массив данных типа UA_Argument (стандартный тип OPC UA).

На один и тот же узел MethodNode могут ссылаться несколько объектов и типов объектов. В связи с этим запрос сервиса Call содержит идентификаторы NodeId и для вызываемого метода, и для объекта, предоставляющего контекст.

Следующие два обязательных атрибута с типом данных UA_Boolean специфичны для класса узлов MethodNode: executable, userExecutable. Они показывают, доступен ли метод для вызова в текущий момент.

2.2.6.8. Представления (ViewNode)

Каждое представление определяет подмножество узлов в адресном пространстве. Представления можно использовать при просмотре информационной модели, чтобы сосредоточиться только на интересующем пользователей подмножестве узлов и ссылок. Узлы класса ViewNodes можно создавать и взаимодействовать с ними, но их просмотр с помощью сервиса Browse в настоящее время не поддерживается в open62541.

2.2.7. Идентификация узлов в адресном пространстве. Тип данных UA_NodeId. Пространства имен (NameSpases)

Каждый узел в адресном пространстве имеет уникальный идентификатор в рамках сервера. Идентификаторы имеют стандартный тип данных UA_NodeId. Приведем определение этого типа из файла open62541/types.h:

```
enum UA_NodeIdType {
UA_NODEIDTYPE_NUMERIC   = 0, /* In the binary encoding, this can also become 1 or 2 (two-byte and four-byte  encoding of small numeric nodeids) */
    UA_NODEIDTYPE_STRING        = 3,
    UA_NODEIDTYPE_GUID          = 4,
    UA_NODEIDTYPE_BYTESTRING = 5
};
typedef struct {
    UA_UInt16 namespaceIndex;
    enum UA_NodeIdType identifierType;
    union {
        UA_UInt32    numeric;
        UA_String    string;
        UA_Guid      guid;
        UA_ByteString byteString;
```

} identifier;} UA_NodeId;

Поле namespaceIndex имеет следующий смысл. С каждым экземпляром сервера связано не менее двух пространств имен, из которых могут выбираться идентификаторы. URI этих пространств регистрируются в массиве NamespaceArray. В пространстве имен каждое имя уникально. Одинаковые имена из разных пространств должны относиться к разным узлам. Поле namespaceIndex указывает индекс пространства имен идентификатора в массиве NamespaceArray. Индекс 0 зарезервирован для предопределенного пространства имен от OPC Foundation (далее – «системное пространство имен». Индекс 1 определяет пространство имен сервера. Существуют и другие пространства имен, которые могут быть зарегистрированы в массиве.

В файле open62541/types.h определены static UA_INLINE - функции для упрощения создания идентификатора узла:

- static UA_INLINE UA_NodeId UA_NODEID_NUMERIC(UA_UInt16 nsIndex, UA_UInt32 identifier);
- static UA_INLINE UA_NodeId UA_NODEID_STRING(UA_UInt16 nsIndex, char *chars).

В файле open62541/ns0ids.h приведены идентификаторы всех предопределенных узлов адресного пространства.

2.3 Концепция коммуникационного протокола OPC UA

2.3.1 Транспортный уровень

Стандарт OPC UA определяет коммуникацию на основе транспортных механизмов TCP, HTTP и SOAP, определенную в стандарте. Программная реализация open62541 использует двоичный протоколе на основе TCP, поскольку он является наиболее распространенным транспортным уровнем для OPC UA.

TCP-соединение открывается для соответствующего имени хоста и порта. При открытии соединения устанавливаются основные настройки соединения, такие как максимальная длина сообщения.

2.3.2 Протокол «клиент/сервер»

В OPC UA определены два способа обмена данными между серверами и клиентами: «клиент/сервер» (Client/Server) и «Издатель/ Подписчик» (Pub/Sub). Мы ограничимся рассмотрением классической технологии «клиент/сервер», основанной на принципе «Запрос/Ответ». Клиенты отправляют запросы на сервер. Сервер отправляет ответы только на соответствующие запросы.

Протокол OPC UA допускает асинхронные ответы. Сервер не должен немедленно отвечать на запросы, и ответы могут отправляться в другом порядке. Такая необходимость возникает в случаях, когда требуется значимое время для обработки запроса (например, при вызове метода или при считывании данных с датчика с задержкой). Для подобных случаев стандарт OPC UA определяет в рамках клиент-серверного протокола технологию «подписки» (Subscrip-tions). Подписки реализуются с помощью специальных запросов, где ответ задерживается до публикации уведомления.

Примечание – Технологию подписки, поддерживаемую в рамках клиент-серверного протокола OPC UA, следует отличать от технологии PubSub, определенной в части 14 стандарта. PubSub – это расширение концепции подписки с целью интеграции коммуникации «многие ко многим» с OPC UA. PubSub не использует протокол клиент-сервер. Подписки реализуют индивидуальную связь клиента с сервером.

Клиент-серверное соединение для двоичного протокола OPC UA состоит из трех вложенных уровней: TCP-соединение с сохранением состояния, безопасный канал связи (Secure-Channel), сеанс связи (Session, далее – «сессия»).

2.3.3 Настройки безопасности. Конечные точки (endpoint)

Чтобы подключиться к серверу, клиенту необходимо знать сетевой адрес, протокол и настройки безопасности.

Информация, необходимая для установления соединения между клиентом и сервером, хранится в так называемой конечной точке сервера (endpoint). Сервер может предоставить несколько конечных точек. Конечная точка представляет собой набор следующих данных:

- URL-адрес конечной точки (протокол и сетевой адрес);
- политика безопасности (название набора алгоритмов безопасности и длина ключа шифрования);
- SecurityMode (уровень безопасности для обмена сообщениями);
- тип аутентификации пользователя.

Сервис GetEndpoints возвращает список доступных конечных точек. Этот сервис можно вызвать без открытия сессии и без шифрования сообщений. Это позволяет клиентам сначала обнаружить доступные конечные точки, а затем использовать соответствующую политику безопасности, которая может потребоваться для открытия сессии.

2.3.4 Безопасный канал связи (SecureChannel)

Безопасные каналы SecureChannels создаются над TCP-соединением. Канал SecureChannel устанавливается по запросу OpenSecure-Channel.

Безопасный канал характеризуется уровнем безопасности передачи сообщений (SecurityMode) и политикой безопасности (Security Policy).

Определены три допустимых уровня безопасности для канала SecureChannel: None, Sign, SignAndEncrypt.

При уровнях безопасности с подписью или шифрованием (Sign или SignAndEncrypt) сообщения шифруются с использованием асимметричного алгоритма шифрования (криптография с открытым ключом). В рамках сервиса OpenSecureChannel клиент и сервер устанавливают общий секрет по изначально незащищенному каналу. Для последующих сообщений общий секрет используется для симметричного шифрования, которое выполняется намного быстрее.

Различные политики безопасности, специфицированные в части 7 стандарта OPC UA, определяют алгоритмы для асимметричного и симметричного шифрования, длины ключей шифрования, хэш-функции для подписи сообщений и т. д. Примеры политик безопасности: None, Basic256Sha256.

2.3.5 Сессия (Session)

Сессия создается над каналом SecureChannel. Сессия гарантирует пользователям возможность аутентификации без отправки своих учетных данных в открытом виде.

Определены следующие механизмы аутентификации: анонимный вход, имя пользователя/пароль, сертификаты Kerberos и x509.

Для установления сессии используются два сервиса: CreateSession и ActivateSession. Сервис ActivateSession может использоваться для переключения существующей сессии на другой канал SecureChannel. Такое переключение позволяет повторно использовать существующую сессию при разрыве соединения.

2.3.6 Структура сообщения

Сообщение протокола OPC UA состоит из заголовка и тела.

Структура заголовка одинакова для всех сообщений и определяется стандартным типом данных OPC UA. Заголовок сообщения содержит базовую информацию, включая длину сообщения, идентификаторы сессии и канала SecureChannel, а также данные для связи запроса с соответствующим ответом. Специальное поле «Chunking» («разбиение на фрагменты») определяет способ разделения и повторной сборки сообщений, длина которых превышает максимальный размер сетевого пакета.

Структура тела сообщения зависит от того, какой сервис его передает, и от того, является ли оно запросом или ответом. Для каждого сервиса в системе типов OPC UA определены типы данных, соответствующие его запросу и ответу. Текст сообщения начинается с идентификатора типа данных, затем следует основная полезная нагрузка сообщения.

Примечание – Определения структур и идентификаторов типов данных, соответствующих запросам и ответам сервисов, содержатся в файле open62541/types_generated.h.

3 Общая характеристика БПП OPC UA

3.1 Стек OPC UA

Стек БПП OPC UA унаследован от библиотеки open62541 и обладает следующими характеристиками [4]:

- транспортные механизмы реализованы в соответствии с профилем UA-TCP UA-SCUA-Binary. Этот профиль определяет комбинацию сетевого протокола, протокола безопасности и кодирования сообщений, оптимизированную для низкого потребления ресурсов и высокой производительности. Он включает в себя сетевой протокол на основе TCP UA-TCP 1.0, бинарный протокол безопасности UA-SecureConversation 1.0 и бинарное кодирование сообщений UA-Binary 1.0;

- поддерживаются следующие виды шифрования сообщений: None, Basic128Rsa15, Basic256, Basic256Sha 256, Aes128Sha256-RsaOaep;

- поддерживаются следующие виды аутентификации: Anonymous, User Name Password, X509 Certificate.

3.2 SDK для разработки серверных приложений

В БПП OPC UA поддерживаются следующие возможности сервера:

- поддержка всех типов узлов OPC UA;

- контроль доступа для отдельных узлов;

- поддержка создания информационных моделей на стороне сервера на основе стандартных определений XML (наборов узлов);

- поддержка добавления и удаления узлов и ссылок во время выполнения;

- поддержка наследования и создания экземпляров типов объектов и переменных (пользовательский конструктор/деструктор, создание экземпляров дочерних узлов);

- поддержка подписок (Subscriptions)/

контролируемых элементов (MonitoredItems) (уведомления при изменении данных).

В дистрибутив БПП OPC UA входит пример сервера (server_ctt), созданный с использованием open62541 v1.0, реализованный в соответствии с профилем OPC Foundation «Micro Embedded Device Server» ([2]: Part 7: Profiles). Профиль поддерживает следующие возможности, специфицированные в стандарте OPC UA: связь клиент/сервер, подписки, вызовы методов и безопасность с политиками безопасности «Basic128Rsa15», «Basic256», «Basic256-Sha256», а также вызов методов и управление узлами.

3.3 SDK для разработки клиентских приложений

SDK для разработки клиентских приложений позволяет реализовывать следующие сервисы:

- DiscoveryServiceSet – группа сервисов обнаружения доступных серверов в составе: FindServers, GetEndpoints,RegisterServer;

- SecureChannelServiceSet – группа сервисов формирования безопасного канала в составе: OpenSecureChannel, CloseSecureChannel;

- SessionServiceSet – группа сервисов формирования сессии в составе: Session Service Set, CloseSession, .ActivateSession;

- Node Management Service Set – группа сервисов управления узлами в составе: AddNodes, AddReferences, DeleteNodes, DeleteReferences;

- ViewServiceSet – группа сервисов работы с представлениямивсоставе: BrowseNext, TranslateBrowsePathsToNodeIds, RegisterNodes, UnregisterNodes;

- AttributeServiceSet – группа сервисов управления атрибутами в составе: Read,Write;

- MethodServiceSet-группа сервисов вызова методов в составе: Call;

- MonitoredItemsServiceSet – группа сервисов для работы с отслеживаемыми объектами (позволяет клиенту создавать отслеживаемые объекты и привязывать их к переменным, атрибутам или событиям) в составе: Create-MonitoredItems, DeleteMonitoredItems, Modify-MonitoredItems, SetMonitoringMode, SetTrig-gering;

- Subscription Service Set – группа сервисов управления подписками в составе: CreateSubscription, ModifySubscription, Set-Publishing-Mode, Publish, Republish, DeleteSubscriptions, TransferSubscriptions.

Примечание – Выше перечислены только те сервисы из состава библиотеки open62541, для которых на текущий момент подтверждена работоспособность в составе БПП OPC UA.

4 Описание модельной задачи

Задача выполняется на программируемом логическом контроллере, содержащем в своем составе процессорный модуль и два модуля ввода-вывода: для дискретных и аналоговых сигналов соответственно. Модули ввода-вывода соединены с процессорным модулем шиной Modbus. Процессорный модуль функционирует под управлением операционной системы ОСРВ Багет 2.7.

На процессорном модуле функционирует сервер OPC UA, разработанный с помощью БПП OPC UA. В адресном пространстве сервера создано два объекта. Каждый объект содержит по 4 переменных. Значения переменных обновляются с частотой 100 мсек. Объект 1 (DI) принимает данные с модуля дискретных сигналов, объект 2 (AI) – с модуля аналоговых сигналов.

Данные объекта 1 получает OPC UA -клиент, функционирующий на инструментальной ЭВМ с микропроцессорной архитектурой x86 под управлением операционной системы семейства Linux. В качестве клиента OPC UA используется свободно распространяемая программа UAExpert с графическим интерфейсом (разработка фирмы UnifiedAutomation). Клиент получает данные с ПЛК путем синхронных запросов по технологии «клиент-сервер».

Данные объекта 2 получает OPC UA -клиент, разработанный на основе Библиотеки OPC UA, функционирующий локально на процессорном модуле ПЛК. Клиент взаимодействует с сервером OPC UA по протоколу TCP. Используется соединение по защищенному каналу с режимом подписи и шифрования. Самоподписанный сертификат и закрытый ключ шифрования сгенерированы посредством программы Open-SSL. Клиент получает данные по подписке.

Отображение адресного пространства сервера в окне программы UAExpert приведено на рис. 3.

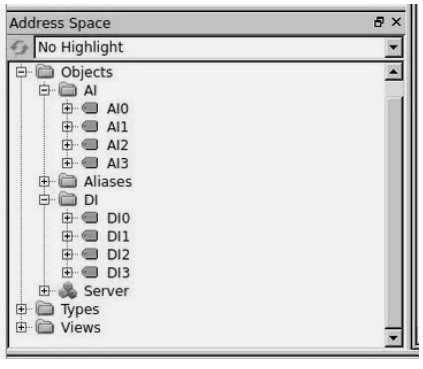

Рис.3 Отображение адресного пространства сервера в окне программы UAExpert.

Ключевые фрагменты кода, использованные при разработке сервера на основе БПП OPC UA, прокомментированы в разделе 8 настоящей статьи.

Примечание – Применение сервисов Subscriptions, MonitoredItems реализовано в клиентской части приложения и не комментируется в настоящей статье.

5 Ключевые фрагменты кода модельной задачи

5.1 Структура серверного приложения

Разработано серверное приложение на основе API БПП OPC UA, которое:

- конфигурирует и запускает сервер OPC UA;
- создает объекты и переменные адресного пространства;
- периодически обновляет в реальном времени значения переменных.

За основу был взят исходный текст примера examples/server_inheritance.c из дистрибутива БПП OPC UA.

Программа, реализующая серверное приложение, структурирована следующим образом.

Головная функция int main(void):

- создает и инициализирует конфигурацию сервера в соответствии с потребностями задачи;
- вызывает пользовательскую функцию createCustomInheritance, которая создает информационную модель задачи в адресном пространстве сервера;
- привязывает к конфигурации сервера пользовательскую Callback-функцию testCallback, которая устанавливает связь переменных с внешними источниками данных;
- запускает сервер;
- по завершении работы сервера освобождает ресурсы.

5.2 Конфигурирование сервера

Конфигурационные данные сервера OPC UA (далее – «конфигурация») используются при инициализации сервера. Структура конфигурации определена в заголовочном файле open62541/server.h: struct UA_ServerConfig {}.

Примеры полей структуры UA_ServerConfig:

UA_ApplicationDescription applicationDescription;

– описание приложения;

UA_ByteString serverCertificate;

– сертификат сервера,

const UA_DataTypeArray *customDataTypes;

– массив пользовательских типов данных;

size_t securityPoliciesSize;

UA_SecurityPolicy* securityPolicies;

– массив URI поддерживаемых политик безопасности;

size_t endpointsSize;

UA_EndpointDescription *endpoints;

– список конечных точек;

UA_UInt16 maxSecureChannels;

UA_UInt32 maxSecurityTokenLifetime; // in ms

– ограничения для безопасных каналов;

UA_UInt16 maxSessions;

UA_Double maxSessionTimeout; /* in ms */

– ограничения для сессий.

Традиционно применяется следующий способ создания конфигурации сервера:

- сначала создать структуру конфигурации и инициализировать ее по умолчанию;
- модифицировать те поля, которые важны в конкретной задаче (например, добавить сертификат сервера);
- инициализировать сервер с использо-ванием отредактированной конфигурации.

В головной функции main() серверного приложения конфигурация сервера сформирована в результате следующих действий:

1) определены и инициализированы нулями структуры данных для сертификата, закрытого ключа и для конфигурации сервера: UA_ByteString certificate, UA_ByteString privateKey, UA_ServerConfig config_, *config=&config_;

2) объявлен и инициализирован статический массив реквизитов доступа static UA_UsernamePasswordLogin logins[3];

3) сертификат и закрытый ключ загружены из файловой системы tar;

4) структура конфигурации сервера заполнена значениями по умолчанию с добавлением сертификата, закрытого ключа и списка реквизитов доступа посредством API-функции UA_ServerConfig_setDefaultWithSecurityPolicies (). Списки доверенных и отзываемых сертификатов оставлены пустыми:

```
retval =
UA_ServerConfig_setDefaultWithSecurityPolicies(config, 4840,
&certificate, &privateKey,
trustList, trustListSize,
issuerList, issuerListSize,
revocationList, revocationListSize);
```

(4840 – порт, используемый по умолчанию для TCP-соединения сервера OPC UA);

5) в поле Description URI приложения приведено в соответствие с данными сертификата:

```
config->applicationDescription.applicationUri
=
UA_String_fromChars("urn:open62541.server.application");
```

6) установлен режим пользовательского доступа по логину и паролю (вызов API-функции UA_AccessControl_default). Список пользователей и паролей указан в массиве logins, передаваемом среди аргументов функции:

```
config->accessControl.clear(&config->
```

accessControl);
retval = UA_AccessControl_default(config, false, NULL,&config->securityPolicies[config->
securityPoliciesSize-1].policyUri, 3, logins);
7) создан экземпляр сервера с отредактированной конфигурацией:
UA_Server *server =
UA_Server_newWithConfig(config);

5.3 Создание информационной модели в адресном пространстве сервера

Функция static void createCustomInheritance (UA_Server *server) создает в адресном пространстве следующие узлы:

- объекты типа «папка» с именами DI, AI как компоненты системной папки Objectcs;
- по четыре компонента каждого объекта: переменные с именами DI0, DI1,DI2, DI3 с типом данных UA_BOOLEAN (AI0, AI1,AI2, AI3 с типом данных UA_INT32) с доступом по чтению и записи;
- ссылки типа «hasComponent» между объектами и их компонентами.

Для создания объектов и переменных вызываются функции:

UA_Server_addObjectNode();
UA_Server_addVariableNode().

При выполнении этих функции создаются соответствующие ссылки.

Вызову функций, создающих узел, предшествует создание и инициализация структур, определяющих атрибуты узла:

UA_ObjectTypeAttributes otAttr;
UA_ObjectAttributes oAttr;
UA_VariableAttributes vAttr;

Создается структура для идентификатора будущего узла:

UA_NodeId df_id;

Структуры атрибутов будущего узла инициализируются значениями по умолчанию, затем редактируются поля «description» и «displayName», например:

oAttr = UA_ObjectAttributes_default;
oAttr.description =
UA_LOCALIZEDTEXT("en-US", "DI");
oAttr.displayName =
UA_LOCALIZEDTEXT("en-US", "DI");

Для переменных также редактируются атрибуты доступа и типа данных, например:

vAttr.accessLevel =
UA_ACCESSLEVELMASK_READ |
UA_ACCESSLEVELMASK_WRITE;
vAttr.dataType
=UA_TYPES[UA_TYPES_UINT32].typeId;

Рассмотрим параметры функции UA_Server_addObjectNode() на примере ее вызова для создания объекта DI.

Интерфейс функции:

UA_StatusCode UA_Server_addObjectNode
(
UA_Server *server,
constUA_NodeIdrequestedNewNodeId,
constUA_NodeIdparentNodeId,
constUA_NodeIdreferenceTypeId,
constUA_QualifiedNamebrowseName,
constUA_NodeIdtypeDefinition,
constUA_ObjectAttributesattr,
void *nodeContext,
UA_NodeId *outNewNodeId
)

Параметры:

- UA_Server *server – сервер,
- constUA_NodeIdrequestedNewNodeId – идентификатор создаваемого узла. Идентификатор можно указать явно или запросить новое значение у сервера. Если в качестве параметра указать числовой NodeId с числовым идентификатором 0, то будет выбран случайный свободный числовой NodeId в соответствующем пространстве имен. В примере: UA_NODEID_NUMERIC(1,0) – запрос на выбор свободного числового значения в пространстве имен сервера;
- constUA_NodeIdparentNodeId –идентификатор родительского узла. В примере: UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER) – числовой идентификатор папки Objects в системном пространстве имен;
- constUA_NodeIdreferenceTypeId – идентификатор типа ссылки родительского узла на создаваемый узел. В примере: UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT) – числовой идентификатор типа ссылки « hasComponent» в системном пространстве имен;
- constUA_QualifiedNamebrowseName – атрибут « browseName» создаваемого узла. В примере: UA_QUALIFIEDNAME(1, "DI") – имя "DI" в пространстве имен сервера;

constUA_NodeIdtypeDefinition – идентификатор типа создаваемого объекта. В примере:

UA_NODEID_NUMERIC(0,UA_NS0ID_FOLDERTYPE) – идентификатор стандартного типа объектов «папка» в системном пространстве имен;

- constUA_ObjectAttributes attr – атрибуты создаваемого узла. В примере передается ранее созданная, инициализированная и отредактированная структура oAttr;
- void *nodeContext – контекст узла (пользовательские данные, предназначенные для экспонирования в адресном пространстве). В примере

передается значение NULL;

- UA_NodeId *outNewNodeId - указатель на структуру идентификатора созданного узла. Идентификатор будет записан по этому указателю. В примере передается указатель на ранее созданную структуру (&df_id).

Параметры функции UA_Server_addVariableNode(), использованной в примере для создания узлов переменных, не отличаются по составу и смыслу от параметров функции UA_Server_addObjectNode(). Отметим только отличия в способе их задания при вызове функции в рассматриваемом примере:

- идентификатор создаваемого узла указывался явно как числовой идентификатор в пространстве имен сервера, например: UA_NODEID_NUMERIC(1, 30300);

- в качестве идентификатора родительского узла указывалась структура df_id, в которую был записан идентификатор родительского объекта при его создании;

- в качестве идентификатора типа создаваемого объекта был указан идентификатор базового типа переменной из системного адресного пространства: UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE);

- в качестве указателя на структуру идентификатора созданного узла указан NULL (возвращать идентификатор не нужно, поскольку он задан явно).

5.4 Связь переменных с источниками внешних данных

Сервер OPC UA предоставляет возможность организовать выполнение с заданной периодичностью пользовательских функций с прототипом:

typedef void(*UA_ServerCallback)
(UA_Server*server,void*data);

Это свойство использовано в серверном приложении для организации асинхронного обновления значений переменных адресного пространства сервера дискретными и аналоговыми сигналами, считываемыми с модулей ввода-вывода по шине Modbus.

Разработана пользовательская функция
static void
testCallback(UA_Server *server, void *data),
которая:

- считывает значения сигналов по шине Modbus средствами ОСРВ Багет 2.7;

- преобразует их тип и копирует значения в переменные адресного пространства с использованием функций БПП OPC UA UA_Variant_setScalar(), UA_Server_writeValue().

Головная функция серверного приложения main() содержит вызов функции БПП OPC UA UA_Server_addRepeatedCallback:

UA_Server_addRepeatedCallback(server, testCallback, NULL, 100, NULL);

Этот вызов обеспечивает периодическое выполнение функции testCallback с интервалом 100 мсек.

6 Заключение

Модельная задача продемонстрировала работоспособность библиотеки поддержки протокола OPC UA на отечественном оборудовании под управлением отечественной операционной системы реального времени. Представляется перспективным развивать эту библиотеку. В частности, желательно расширить состав поддерживаемых сервисов, дополнив его сервисами доступа к историческим данным и поддержкой событий.

Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме «Создание и реализация доверенных систем искусственного интеллекта, основанных на новых математических и алгоритмических методах, моделях быстрых вычислений, реализуемых на отечественных вычислительных системах» (FNEF-2024-0001)».

OPC UA Protocol Support Library for Programmable Logic Controllers of the Baguette Family

T. K. Gringauz, D. V. Yarikov

Abstract. The open62541 freely distributed library implementing the OPC UA protocol has been ported to the environment of the domestic real-time operating system of the Baget family. The article provides an overview of the basic concepts of OPC UA, lists the functional capabilities of the ported version of the library, describes a model task, and provides an analysis of key fragments of its code.

Keywords: programmable logic controller, OPC UA protocol, RTOS Baget

Литература

1. А.Н. Годунов, В.А. Солдатов. Операционные системы семейства Багет (сходства, отличия и перспективы). «Программирование», т.40 (2014), № 5, 68 – 76.

2. OPC Unified Architecture Specification. Release 1.04. November 22, 2017. URL: https://reference.opcfoundation.org/ (дата обращения – 31.10.2024).

3. А.А. Титаев. Промышленные сети: учебное пособие, Екатеринбург, издательство Уральского университета, 2020.

4. open62541 Documentation. Release 1.0.5-1-g982f0796. January 05, 2021. URL: https://www.open62541.org/doc/open62541-1.0.pdf (дата обращения – 31.10.2024).

5. open62541 Documentation. Release 1.4. URL: https://www.open62541.org/doc/master/index.html (дата обращения – 31.10.2024).

6. Unified Automation. C++ UA Server SDK Documentation. Release 1.5.2.336. URL: https://documentation.unified-automation.com/uasdkcpp/1.5.2/html/index.html (дата обращения – 31.10.2024).