ЗАО ИТФ «Системы и технологии»

Интерфейс OPC для SCADA-систем

Выполнил: инженер-программист
Огрызков С.А.

Владимир 2001

Содержание

Введение

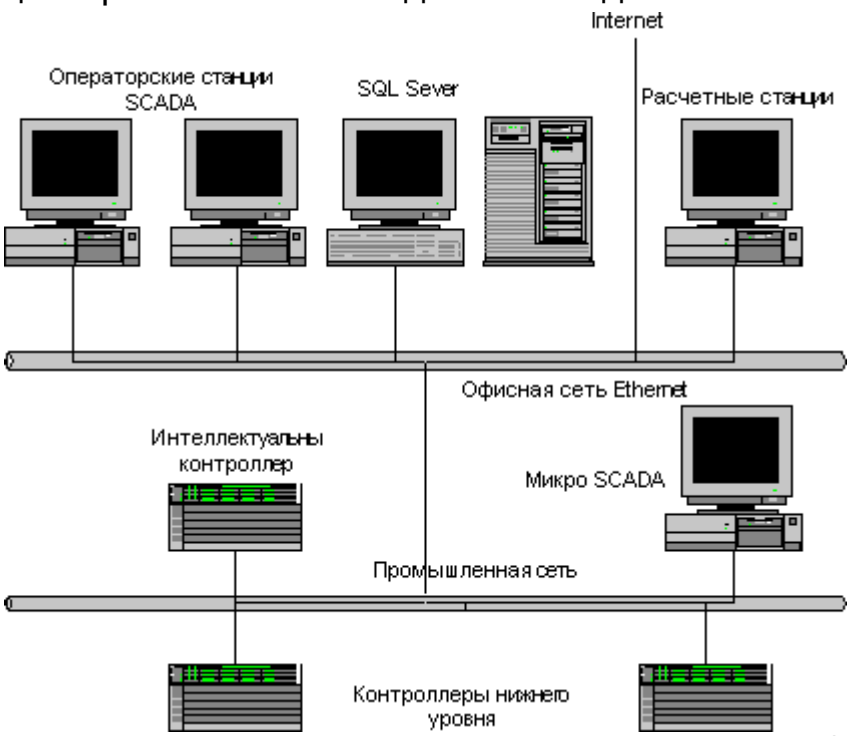
Современная автоматизированная система управления технологическим процессом (АСУТП) представляет собой многоуровневую человеко-машинную систему управления. Создание автоматизированной системы управления сложными технологическими процессами осуществляется с использованием автоматических информационных систем сбора данных и вычислительных комплексов, которые постоянно совершенствуются по мере эволюции технических средств и программного обеспечения.

Концепция *SCADA* (*Supervisory Control And Data Acquisition* – диспетчерское управление и сбор данных) предопределена всем ходом развития систем управления и результатами научно-технического прогресса. Применение SCADA-технологий позволяет достичь высокого уровня автоматизации в решении задач разработки систем управления, сбора, обработки, передачи, хранения и отображения информации.

Открытость SCADA-систем

Перед фирмами-разработчиками систем автоматизации часто встает вопрос о создании собственных (не предусмотренных в рамках SCADA-систем) программных модулей и включение их в создаваемую систему автоматизации. Поэтому вопрос об открытости системы является важной характеристикой системы. Фактически, *открытость системы* означает доступность спецификаций системных (в смысле SCADA) вызовов, реализующих тот или иной системный сервис. Это может быть доступ к графическим функциям, функциям работы с базами данных и т.д.

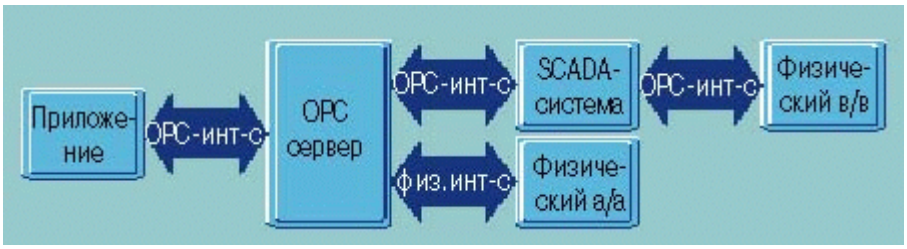
Современные SCADA-системы не ограничивают выбора аппаратуры нижнего уровня, так как предоставляют большой набор драйверов или серверов ввода-вывода и имеют хорошо развитые средства создания собственных программных модулей или драйверов новых устройств нижнего уровня. Сами драйверы разрабатываются с использованием стандартных языков



программирования. Вопрос, однако, в том, достаточно ли только спецификаций доступа к ядру системы, поставляемых фирмой-разработчиком в штатном комплекте, или для создания драйверов необходимы специальные пакеты, или же, вообще, разработку драйвера нужно заказывать у фирмы-разработчика.

Для подсоединения драйверов ввода-вывода к SCADA используются два механизма – стандартный *DDE* (*Dynamic Data Exchange* – динамический обмен данными) и обмен по внутреннему (известному только фирме-разработчику) протоколу. До сих пор *DDE* остается основным механизмом, используемым для связи с внешним миром в SCADA-системах. Но он является не совсем пригодным для обмена информацией в реальном масштабе времени из-за своих ограничений по производительности и надёжности. Взамен *DDE* компания *Microsoft* предложила более эффективное и надёжное средство передачи данных между процессами – *OLE* (*Object Linking and Embedding* – связывание и встраивание объектов).

На базе OLE появился новый стандарт – *OPC* (*OLE for Process Control* – связывание и встраивание объектов для управления процессами), ориентированный на рынок промышленной автоматизации. Новый стандарт, во-первых, позволяет объединять на уровне объектов различные системы управления и контроля, функционирующие в распределённой гетерогенной среде; во-вторых, OPC устраняет необходимость использования различного нестандартного оборудования и соответствующих коммуникационных программных драйверов. С точки зрения SCADA-систем, появление OPC-серверов означает разработку программных стандартов обмена с технологическими устройствами. Поскольку производители полностью разбираются в своих устройствах, то эти спецификации являются для них руководством к разработке соответствующих серверов. Так как эти программные драйверы уже появляются на рынке, разработчики SCADA-систем предлагают свои механизмы связи с OPC-драйверами. OPC-интерфейс допускает различные варианты обмена: получение «сырых» данных с физических устройств, из распределённой системы управления или из любого приложения. На рынке появились инструментальные пакеты для написания OPC-компонентов, например, *OPC-Toolkits* фирмы *FactorySoft Inc.*, включающий *OPC Server Toolkit*, *OPC Client Toolkit* и примеры OPC-программ.



Стандарт OPC

Современные решения в области стандартизации связаны, прежде всего, с фирмой *Microsoft*. Это, в первую очередь, технология *OPC*, то есть *OLE* для технологического управления. Она представляет собой стандартный метод для доступа к периферийным устройствам, SCADA-системам или другим промышленным приложениям, основанным на

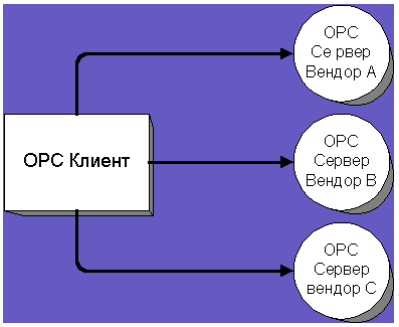
технологиях *OLE*, *COM* (*Component Object Model* – объектная модель компонентов) и *DCOM* (*Distributed COM* – распределённая COM). В общих словах, OPC представлена набором стандартных объектов, методов и свойств, отвечающих требованиям промышленных приложений реального времени. Эти требования включают в себя синтаксис для доступа к объектам, эффективную передачу данных от оборудования к приложениям, способность клиента работать с несколькими серверами одновременно и поддержку конфигурации сервера. Программные пакеты на основе OPC легко интегрировать в бизнес-приложения, поддерживающие OLE.

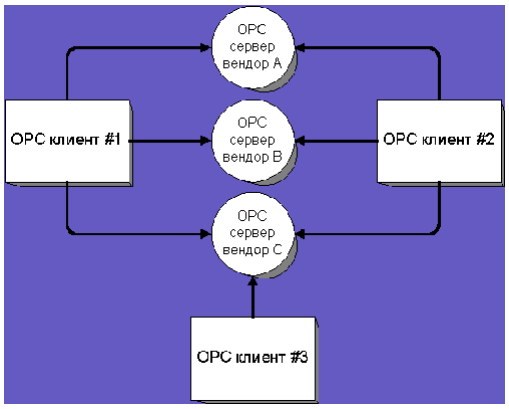
Первая версия OPC вышла в 1995 г. Она не претендовала на стандарт, но была призвана сыграть роль пробного камня для всех заинтересованных сторон. Основной упор был сделан на сбор данных. Более сложные задачи: сигнализация (оповещение о наступлении технологических событий), отслеживание трендов (последовательностей значений параметров, отражающих поведение технологического процесса), моделирование –- были отложены на будущее. В том же 1995 г. появилась независимая некоммерческая организация *OPC Foundation*. Цель её деятельности – централизация управления разработкой нового стандарта. В настоящее время она объединяет около 250 компаний, среди которых *Fisher-Rosemount*, *Rockwell Software*, *Intellution*.

Ниже вкратце будет описан интерфейс доступа к данным по протоколу OPC второй (2.03) версии от 27 июля 1999 года. Замечание: для работы OPC требуется операционная система Windows 95/NT4 и старше.

Общие сведения

OPC-клиент может подключаться как к одному, так и к нескольким OPC-серверам от разных поставщиков.





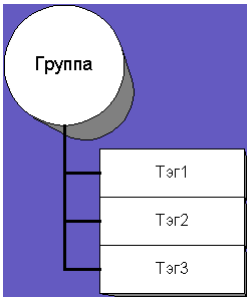
В свою очередь, различные поставщики могут разрабатывать различные OPC-серверы. Код, предоставляемый поставщиком, определяет устройство и данные, к которым каждый сервер имеет доступ, ссылки на данные и детальную информацию о том, как сервер физически обращается к данным.

На верхнем уровне OPC-сервер представляется несколькими объектами: *сервером*, *группой* и *тегом*. Объект *OPC-сервера* содержит информацию о сервере и работает как контейнер для объектов OPC-групп. Объект *OPC-группы* содержит информацию о самом себе и является механизмом для хранения в нём данных, а также логической организации тегов OPC.

OPC-группы предоставляют клиентам возможность организации данных. Например, группа может представляться тегами в соответствующем операторском кадре или отчёте. Данные могут быть считаны или записаны. Соединения, основанные на исключительных ситуациях, также могут быть созданы между клиентом и тегом в группе, который может быть включен или выключен в зависимости от требований. OPC-клиент может определять частоту, с которой OPC-сервер должен предоставлять изменения данных OPC-клиенту.

Возможны два типа групп: *публичные* (общедоступные) или *локальные* (конфиденциальные). Публичные группы необходимы для разделения между множеством клиентов. Локальные группы существуют только для одного клиента. Для публичных групп существуют отдельные интерфейсы.

Внутри каждой группы клиент может определять один или более *OPC-тегов*.



OPC-теги представляют собой связи с источниками данных в сервере. OPC-тег, с точки зрения традиционного интерфейса, недоступен как объект для OPC-клиента. Поэтому для OPC-тега внешние интерфейсы не определены. Доступ к OPC-тегам можно получить через OPC-группу, которая «содержит» OPC-теги.

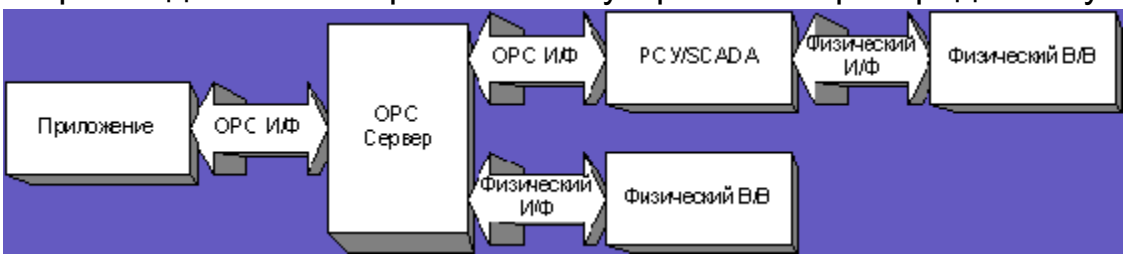
С каждым тегом ассоциированы: значение (*value*), качество (*quality*) и временная метка (*time stamp*). Значение определено в COM-типе *Variant*,

качество аналогично тому, как это сделано в стандарте *Fieldbus*, а временная метка соответствует Windows-типу *FILETIME*.

Особое внимание стоит обратить на то, что теги не являются источниками данных, а лишь связями к ним. Проще всего понять OPC-тег как указатель на адрес с данными, а не как физический источник данных.

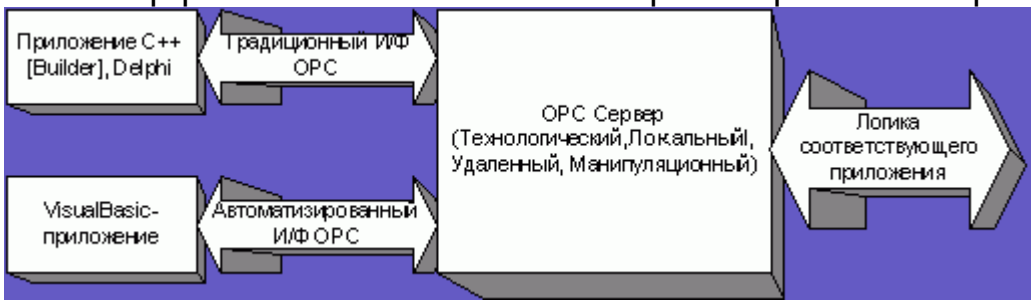
Где нужен OPC

Хотя OPC разработан в первую очередь для получения данных с сетевого сервера, интерфейсы OPC могут быть использованы в самых различных местах приложения. На самом нижнем уровне они могут передавать «сырые» данные из физических устройств в распределённую систему управления (РСУ) или SCADA-систему, или из РСУ или SCADA в приложение. Стандарт делает возможным создание OPC-сервера, который позволяет клиентским приложениям получать доступ к данным многих OPC-серверов от разных поставщиков, работающих на различных узлах (контроллерах, АРМах и т.п.) через единственный объект.



Базовая архитектура и компоненты OPC

Стандарт OPC включает в себя спецификации двух наборов интерфейсов: традиционного OPC-интерфейса (*OPC Custom Interfaces*) и автоматизированного OPC-интерфейса (*OPC Automation Interfaces*). Автоматизированный интерфейс был значительно пересмотрен во второй версии стандарта. Стандарт определяет, что лучшим решением является поддержка обоих типов интерфейсов.



Спецификация OPC включает описание соответствующих COM-интерфейсов. Как и все потомки COM, архитектура OPC – клиент-серверная модель, в которой OPC-сервер, содержащий компоненты, обеспечивает интерфейс доступа к OPC-объектам и управляет ими. Сервер также должен группировать и оптимизировать запросы к данным от различных клиентов, чтобы обеспечивать эффективное взаимодействие с физическим устройством. Данные, возвращаемые устройством, должны буферизироваться для асинхронного распространения или синхронного чтения различными OPC-клиентами. Данные OPC в физических устройствах сервер обновляет по заданию OPC-клиентов.

Ниже будет приведён список доступных объектов, их интерфейсов и методов OPC. Более подробное их описание см. в стандарте OPC.

- **Объект *OPCServer***

  - Интерфейс *IConnectionPointContainer* (появился в версии 2.0)

```
HRESULT EnumConnectionPoints(ppEnum);
HRESULT FindConnectionPoint(riid,ppCF);
```

  - Интерфейс *IOPCBrowseServerAddressSpace* (необязательный)

```
HRESULT QueryOrganization(pNameSpaceType);
HRESULT ChangeBrowsePosition(dwBrowseDirection,szString);
HRESULT BrowseOPCItemIDs(dwBrowseFilterType,szFilterCriteria,
                         vtDataTypeFilter,dwAccessRightsFilter,
                         ppIEnumString);
HRESULT GetItemID(szItemDataID,szItemID);
HRESULT BrowseAccessPaths(szItemID,ppIEnumString);
```

  - Интерфейс *IOPCCommon* (появился в версии 2.0)

```
HRESULT QueryAvailableLocaleIDs(pdwCount,pdwLcid);
HRESULT GetErrorString(dwError,ppString);
HRESULT SetClientName(szName);
```

  - Интерфейс *IOPCItemProperties* (появился в версии 2.0)

```
HRESULT QueryAvailableProperties(szItemID,pdwCount,ppPropertyIDs,
                                 ppDescriptions,ppvtDataTypes);
HRESULT GetItemProperties(szItemID,dwCount,pdwPropertyIDs,ppvData,
                          ppErrors);
HRESULT LookupItemIDs(szItemID,dwCount,pdwPropertyIDs,
                      ppszNewItemIDs,ppErrors);
```

  - Интерфейс *IOPCServer*

```
HRESULT AddGroup(szName,bActive,dwRequestedUpdateRate,
                 hClientGroup,pTimeBias,pPercentDeadband,dwLCID,
                 phServerGroup,pRevisedUpdateRate,riid,ppUnk);
HRESULT GetErrorString(dwError,dwLocale,ppString);
HRESULT GetGroupByName(szName,riid,ppUnk);
HRESULT GetStatus(ppServerStatus);
HRESULT RemoveGroup(hServerGroup,bForce);
HRESULT CreateGroupEnumerator(dwScope,riid,ppUnk);
```

  - Интерфейс *IOPCServerPublicGroups* (необязательный)

```
HRESULT GetPublicGroupByName(szName,riid,ppUnk);
HRESULT RemovePublicGroup(hServerGroup,bForce);
```

  - Интерфейс *IPersistFile*

```
HRESULT IsDirty();
HRESULT Load(pszFileName,dwMode);
HRESULT Save(pszFileName,fRemember);
HRESULT SaveCompleted(pszFileName);
HRESULT GetCurFileName(ppszFileName);
```

- **Объект *OPCGroup***

  - Интерфейс *IConnectionPointContainer* (появился в версии 2.0)
```
HRESULT EnumConnectionPoints(ppEnum);
HRESULT FindConnectionPoint(riid,ppCF);
```

  - Интерфейс *IDataObject* (устаревший, используется для совместимости)
```
HRESULT Dadvise(pFmt,adv,pSnk,pConnection);
HRESULT Dunadvise(Connection);
```

  - Интерфейс *IOPCAsyncIO* (устар., используется для совместимости)
```
HRESULT Read(dwConnection,dwSource,dwCount,phServer,
             pTransactionID,ppErrors);
HRESULT Write(dwConnection,dwCount,phServer,pItemValues,
              pTransactionID,ppErrors);
HRESULT Cancel (dwTransactionID);
HRESULT Refresh(dwConnection,dwSource,pTransactionID);
```

  - Интерфейс *IOPCAsyncIO2* (появился в версии 2.0)
```
HRESULT Read(dwCount,phServer,dwTransactionID,pdwCancelID,
             ppErrors);
HRESULT Write(dwCount,phServer,pItemValues,dwTransactionID,
              pdwCancelID,ppErrors);
HRESULT Cancel2(dwCancelID);
HRESULT Refresh2(dwSource,dwTransactionID,pdwCancelID);
HRESULT SetEnable(bEnable);
HRESULT GetEnable(pbEnable);
```

  - Интерфейс *IOPCGroupStateMgt*
```
HRESULT GetState(pUpdateRate,pActive,ppName,pTimeBias,
                 pPercentDeadband,pLCID,phClientGroup,
                 phServerGroup);
HRESULT SetState(pRequestedUpdateRate,pRevisedUpdateRate,pActive,
                 pTimeBias,pPercentDeadband,pLCID,phClientGroup);
HRESULT SetName(szName);
HRESULT CloneGroup(szName,riid,ppUnk);
```

  - Интерфейс *IOPCItemMgt*
```
HRESULT AddItems(dwCount,pItemArray,ppAddResults,ppErrors);
HRESULT ValidateItems(dwCount,pItemArray,bBlobUpdate,
                      ppValidationResults,ppErrors);
HRESULT RemoveItems(dwCount,phServer,ppErrors);
HRESULT SetActiveState(dwCount,phServer,bActive,ppErrors);
HRESULT SetClientHandles(dwCount,phServer,phClient,ppErrors);
HRESULT SetDatatypes(dwCount,phServer,pRequestedDatatypes,
                     ppErrors);
HRESULT CreateEnumerator(riid,ppUnk);
```

  - Интерфейс *IOPCPublicGroupStateMgt* (необязательный)
```
HRESULT GetState(pPublic);
HRESULT MoveToPublic(void);
```

- Интерфейс *IOPCSyncIO*

```
HRESULT Read(dwSource,dwCount,phServer,ppItemValues,ppErrors);
HRESULT Write(dwCount,phServer,pItemValues,ppErrors);
```

- **Объект *EnumOPCItemAttributes***

  - Интерфейс *IEnumOPCItemAttributes*

```
HRESULT Next(celt,ppItemArray,pceltFetched);
HRESULT Skip(celt);
HRESULT Reset();
HRESULT Clone(ppEnumItemAttributes);
```

- **Традиционный интерфейс (клиентская сторона)**

  - Интерфейс *IAdviseSink* (устаревший, используется для совместимости)

```
void OnDataChange(pFE,pSTM);
```

  - Интерфейс *IOPCDataCallback*

```
HRESULT OnReadComplete(dwTransid,hGroup,hrMasterquality,
                       hrMastererror,dwCount,phClientItems,
                       pvValues,pwQualities,pftTimeStamps,
                       pErrors);
HRESULT OnWriteComplete(dwTransid,hGroup,hrMastererr,dwCount,
                        phClientItems,pErrors);
HRESULT OnCancelComplete(dwTransid,hGroup);
HRESULT OnDataChange(dwTransid,hGroup,hrMasterquality,
                     hrMastererror,dwCount,phClientItems,pvValues,
                     pwQualities,pftTimeStamps,pErrors);
```

  - Интерфейс *IOPCShutdown*

```
void ShutdownRequest(szReason);
```

Список использованных источников

1. Журнал *«Мир компьютерной автоматизации»* (http://www.mka.ru/).

2. Информационный портал по SCADA-системам *SCADA.ru* (http://www.scada.ru/).

3. Официальный сайт *IEEE Computer Society* (http://computer.org/).

4. Официальный сайт комитета *OPC Foundation* (http://www.opcfoundation.org/).

5. Официальный сайт компании *«РТСофт»* (http://www.rtsoft.ru/).

6. Публикации на сайте кафедры молекулярной физики УГТУ (http://www.mp.dpt.ustu.ru/).

7. Сайт *«Промышленная автоматизация в России»* (http://www.industrialauto.ru/).