# OPC-СЕРВЕРЫ С ОТКРЫТОЙ АРХИТЕКТУРОЙ – СРЕДСТВА ВЗАИМОДЕЙСТВИЯ КОМПОНЕНТОВ В ПРОМЫШЛЕННОЙ АВТОМАТИЗАЦИИ

**В.Г. Давыдов, Чыонг Динь Тяу**

**(Санкт-Петербургский государственный политехнический университет)**

*Рассматриваются технологии COM и OPC, затрагиваются алгоритмы и производительность обмена данными между OPC-приложениями. Предлагаются структуры OPC-сервера с открытой архитектурой.*

## Введение

В настоящее время информационная АСУ промышленным предприятием имеет иерархическую структуру, включающую следующие неотъемлемые уровни:

– нижний уровень представляет собой технологические устройства: датчики, контролеры, механизмы и т. д. Поток информации от нижнего уровня должен быть предоставлен пользователям и всем приложениям среднего уровня, использующим их посредством цифровых коммуникационных протоколов связи. При этом в системе не должно возникать проблем несовместимости;

– средний уровень – SCADA-системы для сбора данных и диспетчерского управления ТП на производстве. Этот уровень обеспечивает вторичную обработку данных, которые получены с нижнего уровня, сохранение данных и их доступность приложениям и пользователям верхнего уровня;

*Почему ищут ключ к решению проблемы, а не дверь с замочной скважиной, которую можно открыть этим ключом?*

Журнал "Автоматизация в промышленности"

– верхний уровень – приложения управления ресурсами предприятия. Информация с уровня SCADA-систем должна быть доступной для этого уровня, т.е. доступ к данной информации из прикладных программ не должен вызывать проблем.

Для обеспечения совместимости между уровнями и создания эффективной интегрированной системы управления предприятием системный интегратор или разработчик АСУТП должен извлекать данные ТП в РВ с самого нижнего уровня и построить "прозрачный" путь получаемым данным к самым верхним уровням. Чтобы получить систему, отвечающую всем потребностям заказчика, системному интегратору или разработчику необходимо использовать инструментальные средства управления различных уровней – SCADA-пакеты, БД, электронные таблицы. Ключ к этому – открытая и эффективная коммуникационная архитектура взаимодействия между приложениями, которую предлагает стандарт OPC (OLE for Process Control – основной промышленный стандарт взаимодействия между программными компонентами сбора данных и управления, основанный на Component Object Model (COM) фирмы Microsoft).

## OPC-технология и OPC-приложение

OPC – технология для систем промышленной автоматизации, предназначенная для обеспечения универсального механизма обмена данными между датчиками, исполнительными механизмами, контроллерами, УСО и системами представления технологической информации, оперативного диспетчерского управления, а также СУБД.

OPC-технология создана консорциумом OPC Foundation, в котором участвуют практически все мировые ведущие производители аппаратного оборудования и ПО для промышленной автоматизации. На сегодняшний день OPC-технология в определенной степени реализована и продолжает развиваться. OPC Foundation пытается охватить все аспекты, связанные с взаимодействием между компонентами ПО, между ПО, SCADA-системами и технологическим оборудованием. В настоящее время насчитывается порядка десяти OPC-спецификаций – Data Access (доступ к данным РВ), Alarms & Events (обработка тревог и событий), Historical Data Access (доступ к историческим данным) и т.д. Поэтому можно определить OPC как стандарт взаимодействия между программными компонентами сбора данных и управления, основанный на COM-технологии фирмы Microsoft. Через OPC-интерфейсы одни приложения могут читать или записывать данные в другие приложения, обмениваться событиями, оповещать друг друга о нештатных ситуациях, осуществлять доступ к данным, зарегистрированным в архивах. Эти приложения могут располагаться как на одном

компьютере, так и быть распределенными в сети. При этом независимо от фирмы поставщика OPC-стандарт, признанный и поддерживаемый всеми ведущими фирмами-производителями SCADA-систем и оборудования, обеспечит их совместное функционирование [1].

Популярный класс OPC-приложений представляет собой специализированные OPC-серверы конкретных аппаратных устройств, обеспечивающих предоставление информации о состоянии параметров ТП OPC-клиентам на локальном компьютере или в компьютерной сети. Современные SCADA-системы поддерживают OPC-спецификации, являясь OPC-клиентами. В этом смысле зачастую специализированные OPC-серверы разрабатывают фирмы-производители аппаратных средств нижнего уровня. Спецификации OPC поддерживаются во многих современных SCADA-системах.

### OPC – это совокупность COM-интерфейсов

OPC-технология основана на модели распределенных компонентных объектов DCOM (Distributed COM), представляющей собой сетевое расширение COM. Таким образом, можно утверждать, что OPC – это COM для систем управления (COM for Process Control). Приложение, использующее COM-технологию, реализует информационный обмен с помощью COM-объектов. Каждый COM-объект поддерживает несколько COM-интерфейсов. Клиенты могут получить доступ к услугам COM-объекта только через вызовы операций его интерфейсов. COM-интерфейсом является определенная структура памяти, содержащая массив указателей на все операции интерфейса. COM-технология использует глобальные уникальные идентификаторы (Globally Unique Identifier – GUID) для идентификации практически всех объектов (CLSID – Class Identifier) и интерфейсов (IID – Interface Identifier). Система идентификации GUID разработана для использования в среде распределенных вычислений (DCE – Distributed Computing Environment). Сервис COM базируется на протоколе вызовов удаленных процедур (Remote Procedure Call – RPC) в среде DCE. Все COM-интерфейсы должны наследовать общий, базовый интерфейс, реализованный в виде класса IUnknown. Это единственный интерфейс, о котором знают все объекты (клиенты и серверы). В интерфейсе IUnknown имеется метод IUnknown::AddRef() для увеличения счетчика обращений клиентов к серверу и метод IUnknown::Release() для его уменьшения. В библиотеке интерфейсных функций COM имеется функция CoCreateInstance(), которая, используя идентификаторы-параметры CLSID и IID, создает экземпляр соответствующего компонента для информационного обмена и получает указатель на интерфейс этого экземпляра. COM-технология предоставляет для информационного обмена два важных модуля: заместитель (proxy), расположенный в клиенте, и заглушку (stub), расположенную в сервере. Proxy принимает вызов клиента, кодирует и пакетирует его и передает специальному объекту канала, который ответственен за физическую передачу данных по сети. На стороне сервера пакеты принимает stub, в чьи функции входит распаковка вызовов и передача их серверу. Указанный процесс получил название маршалинга (Marshaling). COM-технология поддерживает стандартный маршалинг, который использует proxy и stub, взаимодействующие друг с другом по протоколу RPC.

Рис. 1 [2] и рис. 2 иллюстрируют в качестве примера последовательность действий при передаче данных клиенту из COM-сервера по запросу клиента. Будем считать, что используемый при этом интерфейс определен в классе IReadData. В соответствии с рис. 1

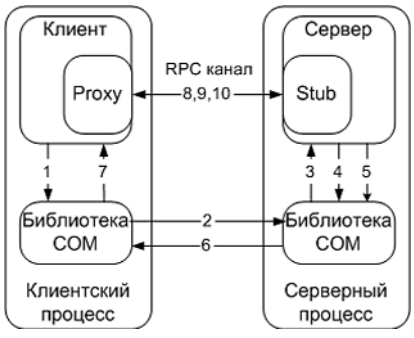


*Рис. 1. COM-архитектура*

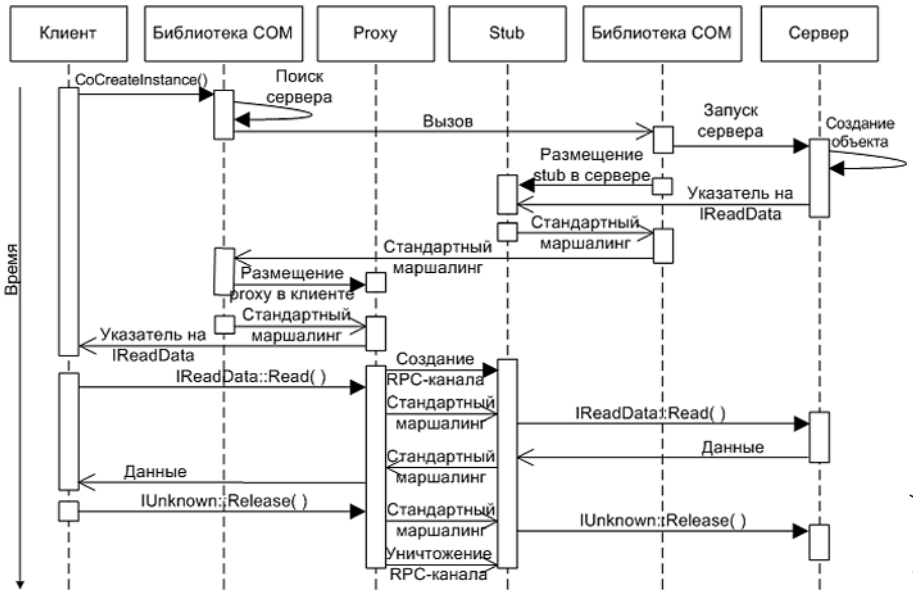


*Рис. 2. Последовательность действий при передаче данных клиенту из COM-сервера*

при чтении выполняются следующие действия:

1. клиент вызывает функцию CoCreateInstance() библиотеки COM с параметрами CLSID сервера и IID интерфейса IReadData;
2. библиотека COM клиентского процесса (в дальнейшем обозначается cCOM) с помощью соответствующей функции определяет местонахождение сервера и передает вызов библиотеке COM серверного процесса (в дальнейшем обозначается sCOM);
3. sCOM запускает сервер;
4. сервер саморегистрируется, создает COM-объект и регистрирует его;
5. sCOM размещает stub в адресном пространстве сервера и извлекает указатель на интерфейс IReadData стандартного маршалинга;
6. sCOM передает необходимую информацию в cCOM;
7. cCOM размещает proxy в адресном пространстве клиента и передает proxy полученный указатель на интерфейс IReadData стандартного маршалинга, который передается клиенту;
8. proxy создает RPC-канал для информационного обмена с stub, возвращает указатель на proxy-интерфейс клиенту. Теперь клиент и сервер могут непосредственно обмениваться информацией через RPC-канал;
9. клиент вызывает IReadData::Read(). Proxy упаковывает вызов и посылает его stub. В свою очередь, stub распаковывает полученный пакет и путем вызова соответствующей функции COM получает требуемую информацию из сервера. Stub упаковывает полученную информацию и посылает упакованный пакет обратно proxy, который, в свою очередь, распаковывает полученный пакет и вручает его клиенту;
10. когда клиент завершает свою работу с севером, он вызывает метод IUnknown::Release(). В результате proxy уведомляет stub о завершении информационного обмена и освобождает RPC-канал (proxy и stub удаляются из адресного пространства клиента и сервера).

Указанную последовательность действий в виде временной диаграммы иллюстрирует рис. 2.

Существуют три типа серверов:

– внутризадачный сервер (in-process) – COM-объекты используют динамически подключаемые библиотеки, сервер работает в адресном пространстве клиента;

– локальный сервер (local) – сервер и COM-объекты работают в отдельном адресном пространстве той же ЭВМ, в которой работает и клиент;

– удаленный сервер (remote) – сервер и его COM-объекты работают на другом компьютере.

В частном случае, когда клиентский и серверный процессы функционируют в одной и той же ЭВМ, но в разных адресных пространствах (локальный сервер), реализация RPC-канала существенно упрощается, но алгоритм информационного обмена сохраняется.

### Обмен данными между OPC-приложениями

Как указывалось выше, в настоящее время существует много OPC-спецификаций, но наиболее широко используется спецификация Data Access по следующим причинам. Во-первых, она обеспечивает доступ к данным в РВ. Во-вторых, эта спецификация появилась раньше других и по этой причине является более востребованной. В силу названных причин, в настоящее время многие SCADA-системы поддерживают только OPC-спецификацию Data Access.

Базовым понятием этой спецификации является элемент данных (Item). Каждый элемент данных имеет значение (Data Value), метку времени (Time Stamp) и признак качества (Quality). Для обмена данными между OPC-клиентом и OPC-сервером в OPC-спецификациях используется тип данных DCOM, называемый VARIANT, т.е. значение Data Value может быть практически любого скалярного типа: булевского, целого, с плавающей точкой или строкой. Признак качества Quality определяет степень достоверности значения, а каждое передаваемое значение сопровождается меткой времени происхождения Time Stamp, показывающей время последнего обновления элемента данных. Time Stamp имеет размер 8 байтов и может показывать UTC-время (всеобщее скоординированное время) с 01.01.1601 с 100-наносекундной точностью.

В общем случае различают OPC-серверы данных, тревог и OPC-серверы архивов. OPC-серверы физических устройств обычно являются только серверами данных. Серверы тревог и архивов чаще всего "паразитируют" на серверах данных. В простых случаях все указанные разновидности OPC-серверов могут быть реализованы в виде одного физического OPC-сервера.

С помощью OPC-серверов данных SCADA-система может обращаться к любому аппаратному устройству путем вызова определенных функций OPC-сервера, подписываться на получение определенных данных какого-либо канала или устройства. В свою очередь OPC-сервер опрашивает аппаратное устройство, вызывает известные функции клиента, подтверждает клиенту получение данных и посылает требуемые клиенту данные.

Обмен данными между OPC-клиентом и OPC-сервером может быть реализован в трех режимах [3] – синхронного и асинхронного чтения/записи и подписки (только чтение).

– При синхронном чтении/записи клиент посылает серверу запрос со списком интересующих его переменных и ждет, когда сервер его выполнит.

– При асинхронном чтении/записи клиент посылает серверу запрос, а сам продолжает работать. Когда сервер выполнил запрос, клиент получает уведомление. Этот режим обладает боль-

*Таблица 1. Производительность OPC-серверов фирмы Intelltion (на компьютере с процессором Pentium 233, ОС Windows NT 4.0 Workstation)*

| Тип сервера | | Число элементов/ транзакции | Кратность передач | Время, с | Затраты процессорного ресурса (%) | Скорость опроса, элемент/с |
|---|---|---|---|---|---|---|
| Внутризадачный | Клиент | 100 | 100000 | 9 | 100 | 1111000 |
| | Сервер | | | | | |
| | Клиент | 1 | 5000000 | 22 | 100 | 227000 |
| | Сервер | | | | | |
| Локальный | Клиент | 100 | 10000 | 16 | 40 | 62500 |
| | Сервер | | | | 60 | |
| | Клиент | 1 | 50000 | 15 | 50 | 3333 |
| | Сервер | | | | 50 | |
| Удаленный (Ethernet 10Base–T) | Клиент | 100 | 1000 | 15 | 10 | 6666 |
| | Сервер | | | | 12 | |
| | Клиент | 1 | 3000 | 9 | 10 | 333 |
| | Сервер | | | | 10 | |

шей гибкостью и рекомендуется в большинстве случаев, т. к. он минимизирует затраты процессорного времени и сетевых ресурсов, особенно при передаче значительно большого числа элементов данных.

– В случае подписки клиент передает серверу список интересующих его переменных, а сервер присылает клиенту информацию об изменившихся переменных из этого списка. При этом сервер будет передавать клиенту информацию только тогда, когда данные изменились, причем эти данные передаются единым блоком. Эти меры позволяют существенно ускорить обмен данными, особенно если речь идет о взаимодействии через сеть.

В зависимости от реализации OPC-сервера, OPC-клиенты могут получать данные либо прямо от устройства (через OPC-сервер), или от кэша OPC-сервера.

Важной характеристикой является производительность OPC-сервера. Производительность OPC-сервера зависит от ряда факторов – вида OPC-сервера (внутризадачный, локальный или удаленный), возможности группировки данных для отправки клиентам, ОС (по результатам исследований центра Doculabs [4] получено, что любое приложение выполняется в среде Windows 2000 быстрее, чем в Windows NT на 16...122 %) и т.д. Сказанное иллюстрирует табл. 1 (результаты тестирования производительности OPC-сервера, разработанного фирмой Intellution [5]).

Из таблицы следует, что передача один раз в 1 с 100 элементов займет значительно меньше ресурсов, чем одного элемента через каждые 10 мс [1].

## Структуры OPC-сервера с открытой архитектурой

Типичная структура реализации OPC-сервера приведена на рис. 3. Клиент может опрашивать данные с помощью функции сервера Read(). Когда клиент желает получить данные из кэша, то функция Read() извлекает требуемые данные прямо из кэша. Клиент может получить свежие данные из устройства и при этом Read() передает управление функции ReadFromDevices(), в свою очередь ReadFromDevices() опрашивает устройство и передает полученные данные клиенту. Функция Write() предназначена для записи данных клиента непосредственно на устройство вывода и/или в кэш данных. Процесс "Обновление устройства и кэша" обеспечивает периодическое обновление кэша данных (ввод) и обновление устройств вывода. При этом периодичность обновления информации задается клиентом. Процесс "Обновление клиентов" служит для передачи данных клиентам в случае асинхронного обмена и подписки.

В настоящее время имеет место следующая негативная ситуация. Существующие OPC-серверы поддерживают в каждом конкретном случае некоторое подмножество аппаратных интерфейсов, причем состав этого подмножества зачастую отражает конкурентные интересы определенных фирм. Вследствие этого, подмножество поддерживаемых аппаратных интерфейсов не является представительным, а аппаратные средства с более высокими ТЭП в конкретных случаях не удается использовать. Поэтому весьма актуальной представляется разработка OPC-серверов с открытой архитектурой. Под открытостью, в данном случае, понимается возможность простого и быстрого включения в OPC-сервер требуемого аппаратного интерфейса как из числа существующих, так и из вновь появляющихся. Работа в этом направлении (методология, технология и ПО) представляется авторам весьма перспективной.

OPC-сервер с открытой архитектурой должен иметь возможность обмена данными с написанной разработчиком аппаратуры динамической библиотекой (DLL). Данная библиотека должна либо содержать весь код, необходимый для управления конкретным устройством в системах автоматического контроля ТП, либо обеспечивать интерфейс с соответствующей библиотекой, поставляемой производителем оборудования. Все детали COM/DCOM при этом реализованы общей частью сервера, предоставленного в виде выполняемого файла. Интерфейс DLL с сервером должен быть прост для

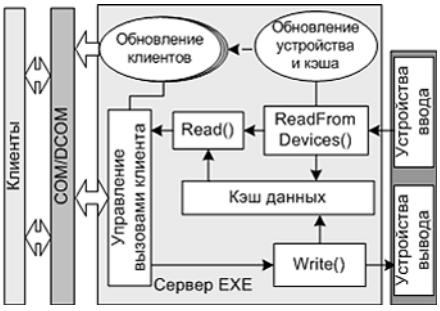


*Рис. 3. Типовая структура OPC-сервера*

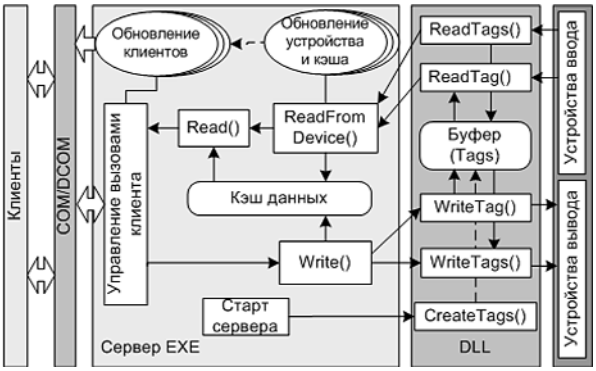


*Рис. 4. Возможная структура OPC-сервера с открытой архитектурой*

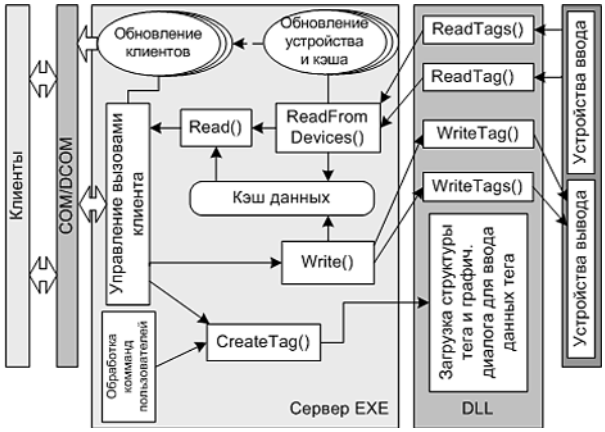


*Рис. 5. OPC-сервер с открытой архитектурой (вариант 2)*

разработчиков. В настоящее время известны несколько OPC-серверов с открытой архитектурой, претендующие на универсальность (наиболее интересен, с этой точки зрения Universal OPC Server фирмы Fastwel). Universal OPC Server прост и недорог, но ему присущ ряд ограничений. Во-первых, клиент получает информацию не непосредственно из устройства, а из внутреннего буфера DLL. При этом запаздывание данных определяется частотой сканирования устройства (для всех групп элементов), которая обеспечивается специальным процессом, размещаемым внутри DLL. Во-вторых, конфигурирование или интерактивное взаимодействие конечного пользователя с устройством или с DLL во время работы сервера не предусматривается.

Для устранения первого из названных недостатков можно использовать структуру OPC-сервера, представленную на рис. 4. Сервер использует DLL со следующими сопровождающими функциями:

– ReadTag() – опрашивает значение одного элемента из устройства или из буфера DLL (Tags), возвращаемое значение функции копируется в кэш данных сервера;

– ReadTags() – циклически вызывается процессом "Обновление устройства и кэша" из сервера EXE (в соответствии с запросами клиентов таких процессов может быть несколько). Она также вызывается, когда клиент реализует ReadFromDevices(). Функция ReadTags() читает данные всех или группы элементов из устройства в буфер DLL. После этого следует многократно вызывать функцию ReadTag(), которая копирует данные из буфера DLL в кэш данных сервера. Если требуется быстрый и без запаздывания доступ к устройству ввода и устройство ввода обладает нужным быстродействием, то вместо функции ReadTags() следует использовать функцию ReadTag(), которая напрямую работает с устройством;

– WriteTag() – записывает значение одного элемента непосредственно на устройство вывода или в буфер DLL;

– WriteTags() – осуществляет запись данных всех или группы элементов из буфера DLL на устройства. Функция циклически вызывается процессом "Обновление устройства и кэша" из сервера EXE. Функция может вызываться, когда происходит завершение одной операции записи клиентом в буфер DLL. Если требуется быстрый вывод и устройство вывода его поддерживает, то следует вместо функции WriteTags() использовать функцию WriteTag();

– CreateTags() – осуществляет создание списка (массива) элементов фиксированного размера в момент запуска сервера EXE. Указанный массив является буфером DLL и его размер рассчитан на наихудший случай. Этот недостаток устраняется в предлагаемой далее второй архитектуре OPC-сервера.

Преимущества предлагаемой структуры по сравнению с Fastwel Universal OPC Server заключаются в следующем. Во-первых, сервер позволяет непосредственно получать информацию из устройства. Во-вторых, предложенная архитектура сервера дает возможность чтения или записи информации для группы параметров (элементов) за одно обращение. В-третьих, процессы "Обновление кэша и устройства" размещаются в сервере EXE и поэтому они открыты для клиентов (клиенты могут динамически менять периоды запуска этих процессов). В-четвертых, сервер EXE поддерживает параллельную работу нескольких процессов "Обновление кэша и устройства" в соответствии с запросами клиентов, что позволяет обновлять данные разных групп с различными периодами.

Однако реализация функции CreateTags() внутри DLL сопровождается рядом недостатков. Во-первых, набор элементов и их число фиксированы и рассчитаны на наихудший случай. При необходимости изменения верхней границы размера буфера или свойств элементов (параметров) нужно модифицировать и перекомпилировать DLL. Во-вторых, использование в DLL буфера приводит к увеличению запаздывания при обмене данными и лишним затратам оперативной памяти.

Для устранения указанных недостатков предложена другая архитектура OPC-сервера (рис. 5). В ней функция CreateTags() удалена из DLL, а в сервер добавлена функция CreateTag(). Кроме того, в DLL буфер не используется, а его роль возложена на кэш данных сервера. При вызове в сервере функции CreateTag() по запросу пользователя с целью со-

здания нового элемента данных в DLL передается информация о типе создаваемого элемента и его характеристики, а характеристики соответствующих устройств ввода/вывода определяются с помощью окна диалога, поддерживаемого в DLL.

Предложенный OPC-сервер с открытой архитектурой (рис. 5) обеспечивает следующие важные возможности:

– подключение аппаратного средства с любым интерфейсом из числа существующих или вновь созданных. Для этого достаточно, базируясь на драйверах или DLL, поставляемых разработчиками аппаратуры, разработать DLL, соответствующую предложенной структуре и функциональным возможностям;

– в интерактивном режиме можно определять нужные свойства элемента данных и/или соответствующего устройства ввода/вывода;

– при работе OPC-сервера набор поддерживаемых элементов данных может динамически меняться, обеспечивая динамическое изменение используемых ресурсов, в частности, памяти.

## Заключение

OPC-технология содержит стандарты, обеспечивающие взаимодействие программных средств промышленной автоматизации. В технологию заложены богатые возможности, которые дают руководителям предприятия возможность интеграции разнородных систем и обеспечивают разработчикам свободу выбора с применением OPC-спецификаций, позволяют не задумываться по поводу поддержки аппаратуры завтрашнего дня. В последнее время консорциум OPC Foundation создал интерфейс, который объединяет все компоненты систем автоматизации и визуализации ведущих производителей. Полагаем, что при дальнейшей доработке этого стандарта и интеграции в него новых функциональных возможностей, OPC станет технологией, которую не сможет проигнорировать ни один производитель средств промышленной автоматизации.

С помощью OPC-технологии разработчик может построить OPC-сервер с открытой архитектурой не только для своей аппаратуры, но и для любых аппаратных устройств нижнего слоя промышленной автоматизации. Чтобы при этом можно было быстро и просто с минимальными издержками включить в арсенал OPC-сервера любой необходимый аппаратный интерфейс и аппаратные модули, требуется еще немало потрудиться в направлении методологии, технологии и разработки ПО.

## Список литературы


1. *Теркель Д.* OLE for Process Control -свобода выбора // Современные технологии автоматизации. 1999. №3.
2. *Chen D., Mok A.K., Nixon M.* Providing real-time support through component object model // Microprocessor and Microsystems 23 (1999).
3. *Куцевич И.В., Григорьев А.Б.* Стандарт OPC — путь к интеграции разнородных систем // Мир компьютерной автоматизации. 2001. № 1.
4. *Pendleton M., Desai G.* @Bench Test Report: Performance and Scalability of Windows 2000, Doculabs, 2000, http://www.duculabs.com
5. DCOM, OPC and Performance Issues, Intellution Inc. 1998. http://www.opcfoundation.org/07_download/performance.doc



***Давыдов Владимир Григорьевич*** *– канд. техн. наук, доцент,*
***Чыонг Динь Тяу*** *– аспирант Санкт-Петербургского государственного политехнического университета.*
*Email: tanchau@mail.ru*