М.В. Грибанов, Д.П. Калачев, С.А. Третьяков

CAL – CAN-протокол прикладного уровня для индустриальных приложений

*Данная статья продолжает цикл публикаций, знакомящих читателей с CAN-технологией. В настоящей публикации описывается один из основополагающих высокоуровневых CAN-протоколов – **CAL** (CAN Application Layer), разработанного и поддерживаемого ассоциацией **CiA** (CAN in Automation). Рассматриваются цели и возможности CAL протокола, а также особенности его практической реализации.*

Распределенные сетевые архитектурные решения, безусловно, должны рассматриваться как высоко-сложные системы. Чтобы гарантировать высокое качество и эффективные результаты при проектировании сетевых систем, степень сложности, которую должен преодолеть проектировщик такой системы, должна быть сведена к приемлемому уровню. Используемые им методологии и архитектурные концепции должны, по возможности, в максимальной степени, скрывать все сетевые детали в программно-аппаратных средствах через поддержку драйверов и сетевых библиотек /1/.

Из-за отсутствия соответствующих методологий и решений при реализации первых сетевых приложений большинство разработчиков программного обеспечения реализовывало информационные процедуры обмена на очень низком уровне (уровне регистров и битовых флажков сетевого контроллера). Такой уровень взаимодействия прикладных задач повышает риск наличия ошибок при низком качестве сетевого программного обеспечения, дает высокую степень индивидуализма программ, а, следовательно, уменьшает возможность его многократного использования и способности к взаимодействию (интероперабельность). Система получается закрытой. Как другой результат – стоимость разработки такой системы довольно высока.

Учитывая, что степень сложности распределенных приложений в будущих проектах будет увеличиваться, существует настоятельная необходимость в сокрытии всех деталей сетевого взаимодействия таким образом, чтобы оградить программиста от всех низкоуровневых манипуляций.

В начале 90-х годов этот вопрос был исследован при выполнении программы ***Prometheus***. Одним из результатов этих исследований является идея ***VLSA*** *(Virtual Level Systems Architecture)* и одно из введенных понятий – ***канал связи*** *(communication link)* /2/.

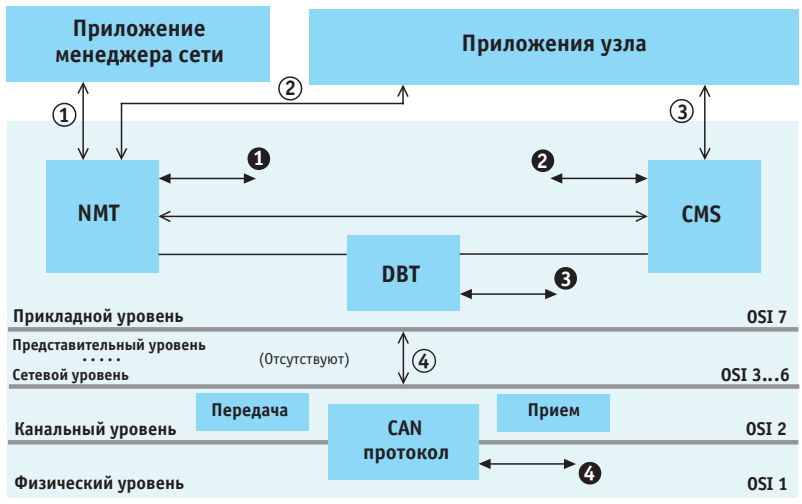
В VLSA-модели канал связи в распределенных системах используется для информационного обмена между переменными. В сущности, эти переменные могут рассматриваться как информационные объекты, расположенные в единой общедоступной памяти. Такая модель является достаточно простой, чтобы понять явления различных методов реализации и взаимодействия, с одной стороны. С другой стороны, такая модель является идеальным средством для упрощения методики проектирования распределенной системы, так как любая транзакция выполняется через операции доступа к переменным.

В 92/93 годах CAN-протокол получил за рубежом большой интерес как дешевый и высокоэффективный протокол связи для промышленных применений. Для того чтобы синхронизировать и стандартизировать различные мнения и подходы, предлагаемые в CAN-системах, в 1992 году была организована ассоциация ***CiA*** /3/. Одной из первых выполненных CiA работ (www.can-cia.de) была разработка открытой концепции, основанной на CAN-сетевой технологии, и определенной как протокол прикладного уровня – ***CAL*** /4/. CAL-спецификацию, которая была опубликована CiA в 1993 году, можно рассматривать как подмножество VLSA-модели.

Спецификация CAL предлагает независимую от приложений объектно-ориентированную среду для реализации распределенных систем, основанных на CAN-сетях. Протокол предлагает объекты и услуги связи, распределения идентификаторов, управления сетью и уровнями. Основные области приложения CAL – распределенные CAN-системы, которые не требуют конфигурируемости и стандартного поведения устройств. Таким образом, CAL спецификация определяет только общие процедуры связи, которые требуются в распределенных системах /5, 6/.

В отличие от протоколов CANopen и DeviceNet, которые также определяют непосредственную структуру и части приложения через фиксированные методы доступа для представления и обмена данными (переменными), CAL не определяет содержимое данных или специфические

Рис.1. Модель CAN/CAL в эталонной модели OSI (без LMT)



Интерфейсы:
① Сервис менеджера сети
② Сервис менеджера узла
③ Сервис обмена сообщениями
④ Сервис канального уровня

Протоколы:
❶ Протокол менеджера сети
❷ Протокол обмена сообщениями
❸ Протокол распределения
❹ CAN протокол

объекты связи, которые должно обеспечивать некоторое устройство или которые ожидаются системой. Используя CAL, пользователь имеет возможность адаптировать систему связи точно к требованиям его приложения или системы. Следовательно, CAL протокол идеально подходит для реализации специфических системных решений подобно медицинским или измерительным системам, а также для реализации закрытых систем управления с децентрализованными интеллектуальными модулями.

Краткая характеристика CAL

Основная цель CAL – предоставление прикладному программисту инструмента для написания программного обеспечения распределенных приложений, использующих CAN для объединения узлов распределенного приложения.

CAL рассматривает программное обеспечение множества устройств, объединенных в сеть, не как слабосвязанные блоки, общающиеся между собой с помощью высокоуровневых сетевых протоколов, а как единое распределенное приложение, различные компоненты которого функционируют на различных узлах сети. Таким образом, CAL предоставляет тот уровень абстракции, который позволяет концептуально связать компоненты программного обеспечения сетевых узлов в единое целое, а также служит хорошей основой для создания программного обеспечения тематических стандартов более высокого уровня.

Многие стандарты промышленных сетевых протоколов не полностью содержат все уровни стандартной модели OSI, и CAL не исключение (рис. 1) /7/.

Причины отсутствия ряда слоев состоят в следующем:

- сама природа CAN не имеет возможности для межсетевой маршрутизации, которую предоставляет ***сетевой уровень*** *(Network Layer)*;
- в распределенных управляющих системах реального времени каждое сообщение передается собственным кадром. CAL обеспечивает транспортировку произвольных данных большого размера на прикладном уровне. Кроме того, CAN протокол уже поддерживает некоторые аспекты надежного подключения. Поэтому ***транспортный уровень*** *(Transport Layer)* также отсутствует;
- в системах управления реального времени не используются связи, ориентированные на сессию. Поэтому ***уровень сессии*** *(Session Layer)* также отсутствует;
- CAL жестко определяет форматы данных для приложений через правила кодирования («Encoding Rules»). Поэтому ***представительный уровень*** *(Presentation Layer)* отсутствует.

Стандарт CAL

Услуги канального уровня

В CAN сообщение передается внутри так называемого ***COB***'а *(Communication Object)*. Каждый COB способен перенести до 8 байт

Рис. 2. Типы сервиса прикладного уровня

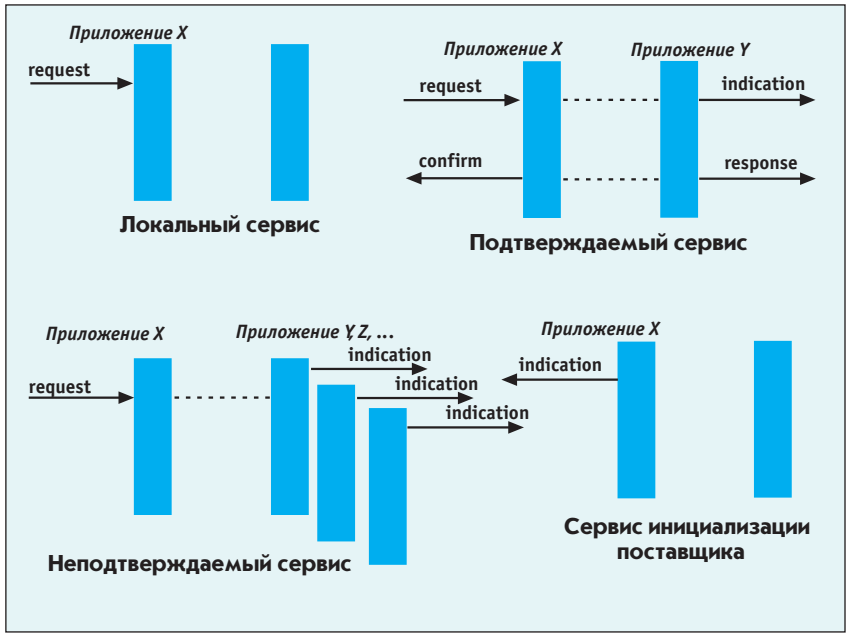
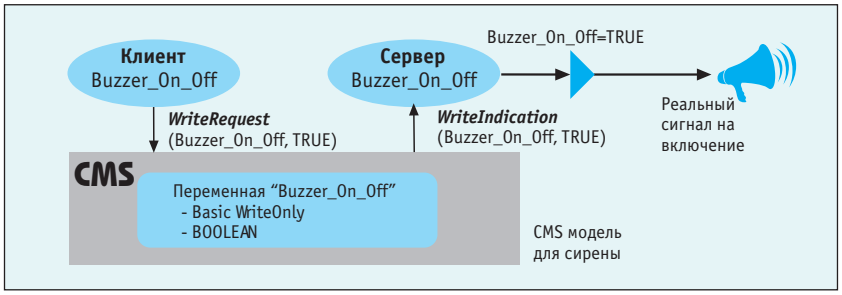


Рис. 3. Пример CMS модели для управления сиреной



данных и характеризуется уникальным идентификатором *(COB-ID)*.

Канальный уровень CAN предлагает примитивы, обеспечивающие широковещательную передачу сообщений или чтение конкретного сообщения по запросу. Таким образом, данные не передаются на определенный узел сети в рамках распределенного приложения (node-oriented network). Каждый узел, используя значение COB-ID, самостоятельно принимает решение, требуются ли ему данные, содержащиеся в принятом пакете (message-oriented network). То есть, вообще говоря, приемник не знает источника принятых данных. Такая архитектура сохраняется и на прикладном уровне (CAL) .

Основные характеристики CAL

Прикладной уровень CAL предоставляет приложению четыре сервисных элемента (service elements):

- ***CMS – CAN Message Specification*** *(элемент сервиса определения сообщений);*
- ***NMT – Network Management*** *(элемент сервиса управления сетью);*
- ***DBT – Distributor*** *(элемент сервиса распределения CAN-идентификаторов);*
- ***LMT – Layer Management*** *(элемент сервиса управления уровнями).*

Протокол CAL определен двояко: с одной стороны определяются достаточно абстрактные сущности прикладного уровня и набор так называемых сервисов, позволяющих оперировать этими сущностями в рамках распределенного приложения (Service Specification), с другой – описываются протоколы взаимодействия этих абстрактных сущностей на уровне описания потоков данных в сети CAN (Protocol Specification) /4/.

Все сервисы CAL реализуются через сервисные примитивы (рис. 2):

- ***request*** – запрос формируется приложением для передачи на уровень CAL;
- ***indication*** – индикация формируется CAL и оповещает приложение о поступлении данных или возникновении некоторого события на уровне CAL;
- ***response*** – ответ формируется приложением для передачи на уровень CAL как реакция на ранее поступившую индикацию;
- ***confirm*** – подтверждение формируется CAL и оповещает приложение о поступлении ответа.

CMS – CAN спецификация сообщений

Сервисный элемент ***CMS*** *(CAN-based Message Specification)* предоставляет собой язык описания и манипуляции с объектами распределенного приложения. Доступ узлов приложения к CMS объектам осуществляется в модели клиент-сервер.

Имеется возможность использовать три типа объектов:

- *переменные;*
- *события;*
- *домены.*

Переменные

Объекты типа ***Переменная*** *(Variable)* обеспечивают передачу данных, инициируемую клиентом. Переменные обеспечивают обмен данными размером не более 8 байт, то есть обмен данными осуществляется с помощью передачи всего одного COB'а.

Примером применения такого объекта может служить тревожная сирена. Сервер объекта – узел, непосредственно управляющий сиреной. Клиент – узел, вырабатывающий управляющий сигнал на включение или выключение. Управление сиреной осуществляется передачей (записью) требуемого булевского значения клиентом серверу (рис. 3).

Для описания каждого объекта стандарт предусматривает сервис ***Define***, определяющий атрибуты объекта (табл. 1).

В зависимости от значений атрибутов ***Class*** и ***Access Type*** переменная может требовать одного COB'а для передачи данных от сервера клиенту – неподтверждаемый сервис (см. рис. 2), либо двух, в том

Рис. 4. Определение CMS объектов

```
/*************************************************************
* Define CMS. Массив дескрипторов объектов
*************************************************************/
CMS_DEFINITIONS = {
  Def_VBWO_S( SIREN_CMD_CANDRV_CHAN, 1005,  BOOLEAN ),
  Def_ES_S   ( AC_STATE_CANDRV_CHAN, 402, STATE_RECORD, 0),
  Def_DM_S   ( AC_DATA_CANDRV_CHAN, 403, 404, UNSIGNED8, 0),
  Def_VBWO_S( AC_CMD_CANDRV_CHAN, 405, CMD_RECORD   ),
  Def_EU_C   ( AC_READ_LDATA_CHAN, 102, LDATA_ITEM   )
};

/*************************************************************
* Define CMS. Индексы CMS объектов в массиве дескрипторов
* объектов
*************************************************************/
enum {
  SIREN_CMD_CMS_HANDLE,
  AC_STATE_CMS_HANDLE,
  AC_DATA_CMS_HANDLE,
  AC_CMD_CMS_HANDLE,
  AC_LDATA_CMS_HANDLE
};
```

случае, если протокол требует запроса и подтверждения (результата) – подтверждаемый сервис. Назначение COB'ов выполняется сервисным элементом DBT.

Стандарт не фиксирует точного синтаксиса сервиса ***Define*** в терминах какого-нибудь языка программирования. Это определяется конкретной реализацией CAL. Например, пакет DμCAL /10/ не предполагает динамического распределения COB'ов (а стандарт это допускает), и при определении переменной следует задавать COB(ы), и нет необходимости задавать имя и приоритет. Для идентификации CMS объектов в приложении удобнее использовать не имя, а индекс в таблице объектов приложения (рис. 4).

Для манипуляции с переменными стандарт предусматривает сервисы чтения ***(Read)***, записи ***(Write)*** и обновления ***(Update)*** данных.

Рис. 5 Задача – сервер Basic WriteOnly переменной , сервис Write

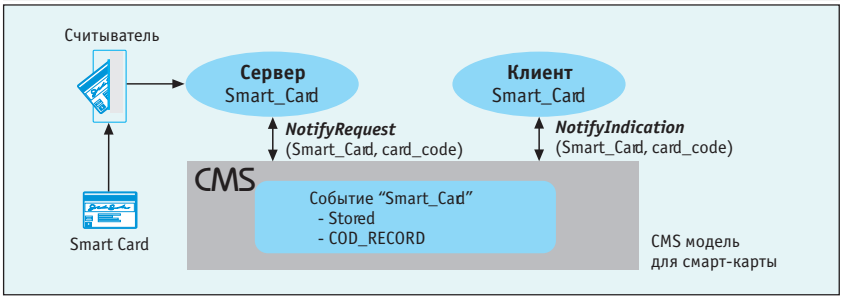
```
//=== Задача управления сиреной
void SirenOnOff_task(void)
{
  BOOL cData;
  Start_VBWO_S( SIREN_CMD_CMS_HANDLE );
  for(;;) {
    Wait_VBWO_S_Indication( SIREN_CMD_CMS_HANDLE,
(byte*)&cData, NOTOUT );
    if( cData ) SirenOn();
    else        SirenOff();
  }//for(;;)

}//SirenOnOff_task
```

На рис. 5 приведен фрагмент программы, реализующий сервер

Таблица 1. Атрибуты CMS-объектов

Name		Каждый CMS объект в рамках одного распределенного приложения имеет уникальное символическое имя, которое, безусловно, должно быть одно и то же на клиентах и серверах объекта. Используется для динамического распределения COB-ID. При статическом распределении COB-ID имя может не использоваться.
User Type		При определении объекта на узле распределенного приложения необходимо указать, кем по отношению к объекту является узел - клиентом или сервером.
Priority		Приоритет COB'а (ов), используемых объектом. Используется при динамическом распределении COB-ID. При статическом распределении COB-ID не используется, т.к. COB-ID'ы задаются при определении объекта.
Inhibit Time		Время запрещения – минимальное время между двумя передачами COB'а. Каждый узел может указать собственное значение. При динамическом распределении COB'ов DBT укажет единое значение для всех узлов, использующих этот объект.
Data Type	Variables и Events	Передаваемые данные могут иметь различный тип. Тип служит для определения размера передаваемых данных и гарантирует одинаковое представление данных на разнотипных узлах.
Error Type	Variables и Events	Подтверждаемые сервисы могут возвращать ошибки. Атрибут определяет тип данных, соответствующих возвращаемому коду ошибки.
Class	Variables	Переменная может относиться к одному из двух классов: базовая (Basic) или мультиплексная (Multiplexed). Для мультиплексной переменной дополнительно должны быть заданы Variable Set Name и Multiplexor. Несколько мультиплексных переменных на одном множестве используют одинаковые COB'ы.
	Events	Событие может относиться к одному из трех классов: неуправляемое (Uncontrolled); управляемое (Controlled), клиент может запретить или разрешить серверу оповещение о событии; событие с памятью (Stored), сервер осуществляет оповещение о событии, кроме того, клиент может прочесть данные события, как для ReadOnly переменной.
	Domains	Домен может относиться к одному из двух классов: базовый (Basic) или мультиплексный (Multiplexed). Мультиплексный домен благодаря различным значениям мультиплексора обеспечивает передачу различных данных некоторого узла, используя один и тот же COB.
Access Type	Variables	Тип доступа клиента к переменной. Допустимые значения:т – только для записи (WriteOnly), для такой переменной клиент может записать новое значение переменной, сервис чтения значения переменной отсутствует – только для чтения (ReadOnly), для такой переменной клиент может прочесть значение переменной, сервис записи значения переменной отсутствует; – для чтения и записи (ReadWrite), клиент может как читать, так и писать значение переменной.
Variable Set Name	Multiplexed Variables	Имя множества, объединяющего несколько мультиплексных переменных. Мультиплексные переменные, принадлежащие одному и тому же множеству, имеют один и тот (те) же COB-ID. Используется при динамическом распределении COB-ID.
Multiplexor	Multiplexed Variables	Мультиплексор (индекс), позволяющий различить мультиплексные переменные на одном множестве.
Multiplexor Data Type	Multiplexed Domains	Для правильной интерпретации мультиплексора на клиенте и сервере следует обеспечить их одинаковое представление, что обеспечивается указанием типа данных мультиплексора.

Рис. 6. Пример CMS модели считывания смарт-карты



Basic WriteOnly переменной для управления сиреной.

Все сервисы CMS, реализуемые через примитив ***indication***, требуют ожидания поступления данных. Если на узле присутствует несколько CMS объектов, которые должны одновременно ожидать приема данных (например, сервер WriteOnly переменной и клиент события), то реализовать такое ожидание удобнее всего в рамках мультизадачной операционной системы. В таком случае реализация, например, сервера одной Write Only переменной будет представлять собой отдельную задачу.

В зависимости от класса и способа доступа для обслуживания сервисов приема/передачи может требоваться один или два COB'а.

События

Объекты типа ***Событие*** *(Event)* обеспечивают передачу данных, инициируемую сервером. События обеспечивают обмен данными размером не более 8 байт.

Примером применения такого объекта может служить датчик считывания смарт-карты, используемой в качестве пропуска (рис. 6). Сервер объекта – узел, содержащий считыватель и оповещающий заинтересованные узлы о прикладывании пропуска. Клиент – узел, принимающий событие. Данные события – код пропуска.

При определении события с помощью сервиса ***Define*** необходимо указать его атрибуты (см. табл. 1).

Для манипуляции с событиями стандарт предусматривает сервисы оповещения ***(Notify)***, чтения ***(Read)***, обновления ***(Store)*** данных и изменения статуса ***(SetControlState)***.

На рис. 7 приведен фрагмент программы, реализующий сервер ***Stored*** события для работы со считывателем пропусков.

При определении события с помощью сервиса Define необходимо указать его атрибуты (см. табл. 1).

Передача больших блоков данных с помощью домена обеспечивается за счет разбиения блока данных на сегменты, помещающиеся в один COB. Таким образом, для сегментированного приема и передачи стандарт предусматривает сервисы ***инициализации передачи*** *(Initiate Domain Download, Initiate Domain Upload)*, собственно ***передачи сегмента*** *(Download Domain Segment, Upload Domain Segment)* и ***прерывания обмена*** *(Abort Domain Transfer)*. Причем если инициатором обмена может выступать только клиент, то прервать обмен может как клиент, так и сервер.

При использовании сегментированного обмена ответственность за

Рис. 7. Фрагменты программы-сервер Stored события, сервис Notify

```
. . .
//=== Инициализация Stored Event - событие модуля доступа
sData.stChange = 0;
Start_ES_S( AC_STATE_CMS_HANDLE, (byte CMS_STORAGE*)&sData );
. . .
//=== Оповещение о событии
sData.stChange = evPersonalPermit;      // Событие - вставка
                                        // карточки личного
                                        // пропуска
memmove( sData.stDat, COD, COD_LEN );   // Копирование кода
                                        // пропуска
StoreAndNotify_ES_S( AC_STATE_CMS_HANDLE,
                              (byte CMS_STORAGE*)&sData );
. . .
```

Домены

Объекты типа ***Домен*** *(Domain)* обеспечивают передачу данных, инициируемую клиентом. Размер массива данных, вообще говоря, не ограничен.

В качестве примера сервера мультиплексного домена можно рассмотреть узел, выполняющий считывание различных данных с некоторого лабораторного оборудования. При этом мультиплексор будет определять то, какие данные требуется получить клиенту (рис. 8).

правильность разбиения домена на сегменты и определение завершения передачи лежит на прикладной программе. Если же применять ***не сегментированный сервис*** *(Domain Download, Domain Upload)*, то все управление обменом выполняет CAL.

На рис. 9 приведен фрагмент программы, реализующий сервер мультиплексированного домена (см. рис. 8).

Типы данных

Сервисы определения переменных и событий содержат атрибут типа данных. CMS определяет стандартные типы данных и способы конструирования производных типов данных. Использование типов позволяет абстрагироваться от внутреннего представления данных в тех или иных процессорах и использовать в рамках одного распределенного приложения системы,

Рис. 8. Пример CMS модели для считывания блока данных

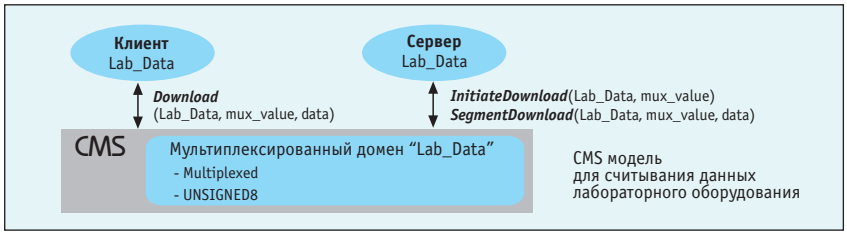


Рис. 9. Задача – сервер Multiplexed домена

```
void DataExchange_task(void){
. . .
Start_DM_S( AC_DATA_CMS_HANDLE );
 for (;;) {
   req = Wait_DM_S_Indication( AC_DATA_CMS_HANDLE, buf,
                                              NOTOUT, state );
   switch (req) {
     case cmsINITDOWN_REQ_DOM://Инициализация загрузки
       switch(mux_value = *bDM_M(buf)){//Анализ мультиплексора
           . . .
      }
      break;
     case cmsDOWN_REQ_DOM://Запрос на очередной сегмент
      switch( mux_value ){
      . . .                          //Прием очередного сегмента
      }//switch( mux_value )
      break;
     case cmsABORT_DOM://Отказ от обмена
      state = dtsINIT;
      continue;
   }//switch( req )
   Send_DM_S( AC_DATA_CMS_HANDLE, buf, (byte*)&mux_value );
 }//for(;;)
}//DataExchange_task
```

различающиеся по внутреннему представлению данных.

Тип данных определяет так называемые ***правила кодирования*** *(Encoding Rules) /4/*. Эти правила четко специфицируют побитовый формат различных типов данных. Однако если в приложении используется формат отличный от того, который определен стандартом CAL, то ответственность за преобразование лежит на приложении. Более того, CAL не имеет достаточных средств контроля того, что для одного и того же объекта на разных узлах используется один и тот же тип данных. Гарантировать это должен прикладной программист, выполняющий интеграцию распределенного приложения.

Реализация DμCAL использует указание типа данных при определении CMS объекта для подключения функций преобразования CAL-приложение для соответствующего типа данных.

NMT – управление сетью

Сервисный элемент ***NMT*** *(Network ManagemenT)* отвечает за собственно сетевые аспекты распределенного приложения. Сервисы уровня NMT не взаимодействуют с прикладными задачами (например, управлением производственным процессом).

Для управления сетью NMT определяет три объекта:

- ***network object*** – сеть, представляет собой набор описаний всех узлов (до 256) сети. Описание сети (т.е. network object) может находиться только на одном узле сети CAN, называемом NMT Master;
- ***node object*** – узел, описание узла сети CAN в network object на NMT Master;
- ***remote node objec***t – удаленный узел, описание узла сети в модуле, модули, содержащие такой объект, называются NMT Slave.

Для каждого ***remote node object*** на NMT Slave должна существовать ***пара node object*** на NMT Master. NMT модель CAL сети представлена на рис. 10.

Основные атрибуты ***node object*** и ***remote node object***: ***NMT Address*** – адрес и ***NodeID*** – номер узла служат для идентификации узла и используются сервисами NMT; ***NodeClass*** – класс узла; ***NodeState*** – состояние узла, определяющее доступные сервисы NMT и возможность выполнения сервисов CMS.

NMT предоставляет три группы сервисов:

- ***Module Control Services*** – сервисы управления модулями обеспечивают определение ***node object*** и ***remote node object***, инициализацию NMT Slave, смену состояния объектов;
- ***Error Control Services*** – сервисы управления ошибками обеспечивают обнаружение и обработку ошибок и отказов CAN сети, удаленные ошибки обнаруживаются через протокол Node/Network Guarding;
- ***Configuration Control Services*** – сервисы управления конфигурированием позволяют конфигурировать узлы, загружать и читать код и данные приложения.

Класс узла определяет набор поддерживаемых узлом сервисов. Минимальный NMT класс соответствует так называемой минимальной NMT функциональности – статическое

Рис. 10. Модель NMT

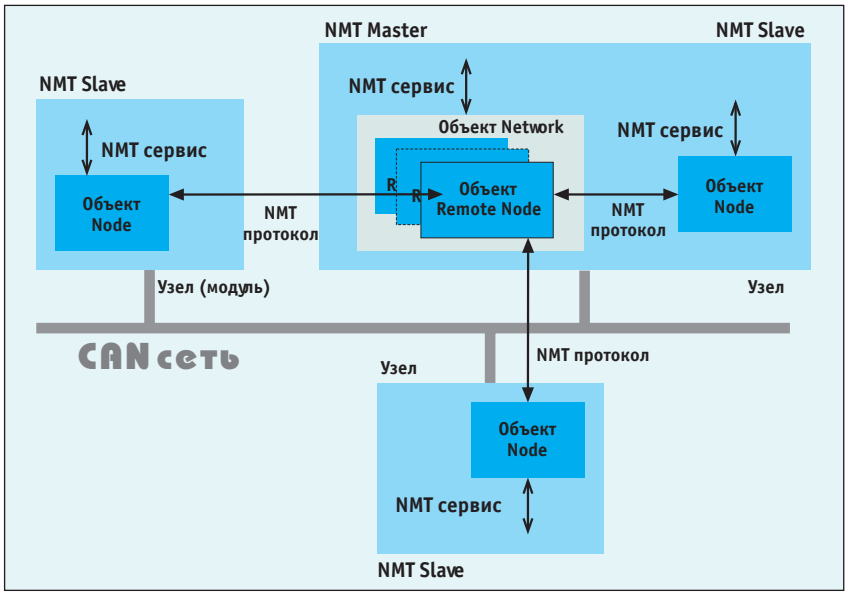
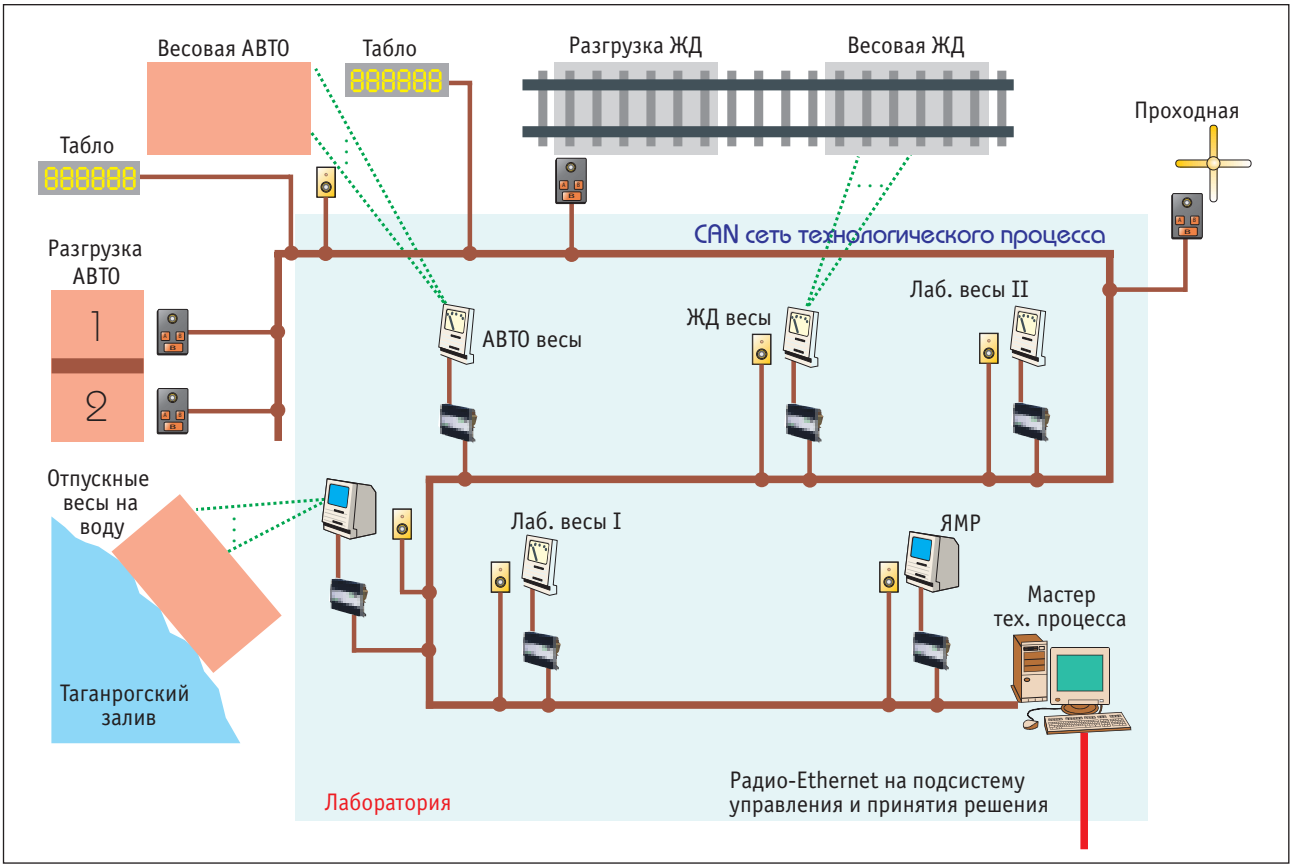


Рис. 11 Структура системы управления приемки зерна на зерновой терминал



назначение NMT параметров и невозможность их изменения средствами CAL.

DBT – распределение идентификаторов

Сервисный элемент ***DBT*** *(DistriBuTor)* обеспечивает динамическое назначение COB-ID'ов и Inhibit Time. Динамическое назначение идентификаторов позволяет осуществлять горячее подключение узлов и оперативную реконфигурацию сети.

Для реализации процесса распределения используется модель Master/Slave, подобная модели NMT (рис. 10). На DBT Master (который может быть только один в сети) размещается COB Database, содержащая информацию, в первую очередь имена всех CMS объектов на всех узлах сети. DBT Slave (совпадающий с NMT Slave) может запросить у DBT Master выполнение распределения идентификаторов.

DBT предоставляет две группы сервисов:

- ***Distribution Control Services*** – сервисы распределения обеспечивают динамическое назначение идентификаторов;
- ***Consistency Control Services*** – сервисы этой группы обеспечивают возможность проверки корректности одноименных CMS объектов, созданных на различных узлах.

LMT – управление уровнями

Сервисный элемент ***LMT*** *(Layer ManagemenT)* позволяет изменять NMT адрес и некоторые параметры ***канального уровня сети*** *(bit timing parameters)*.

Возможность изменения NMT Address обеспечивает выполнение оперативной реконфигурации сети.

Для реализации процесса распределения используется модель Master/Slave, подобная модели NMT (рис. 10). Каждый LMT Slave обладает не изменяемым ***LMT Address*** (серийным номером), благодаря которому LMT Master может идентифицировать LMT Slave, для которого требуется изменить или прочесть параметры.

Пакеты CAL протокола

Существующее на рынке программное обеспечение, реализующее CAL протокол, позволяет легко и быстро создавать CAL приложения. Так, например, пакет ***CAL Protocol Software*** от фирмы IXXAT Automation GmbH (www.ixxat.de), который поставляется в исходном коде, покрывает все коммуникационные задачи, определяемые спецификацией CAL. Это позволяет проектировщику системы сконцентрироваться только на его приложении.

Согласно делению CAL на четыре сервисных элемента (см. рис. 1), CAL Protocol Software также состоит из отдельных модулей. Пакет может быть легко сконфигурирован и адаптирован к функциональным возможностям, требуемым приложением. Это означает, что к прикладным задачам не будут добавлены ненужные функциональные сетевые возможности.

Доступ к CAN-контроллеру (например, передача и прием объектов) выполняется через интерфейс канального уровня данных, который

также реализован в виде отдельного модуля. Это позволяет выполнять простую адаптацию пакета к различным CAN-контроллерам и ресурсам системы. Пакет доступен в версиях Slave и Master/Slave.

В случае, если проектировщик желает полностью отделить прикладную систему от процесса связи по CAN, он может воспользоваться интеллектуальной интерфейсной PC/CAN платой, например iPC-I 320/PCI (на контроллере от DALLAS Semiconductor, совместимом с 8051), iPC-I 165 (на контроллерах от Siemens), tinCAN (PCMCIA), и микрокодом пакета CAL Windows DLL.

Все функциональные возможности CAL протокола реализуются через API, который доступен в виде DLL для Windows 3.11, 95/NT (включая требуемые драйверы). Доступна версия и для DOS, которая может быть легко перенесена на другие операционные системы, подобные VxWorks, VRTX или OS/2. Пакет CAL Windows DLL доступен в версиях Slave и Master/Slave.

Пример реализации распределенной системы с использованием CAL

Все примеры, приведенные в статье, взяты из реального проекта управления приемкой зерна на зерновом терминале (г. Таганрог). Общая структура системы приведена на рис. 11. Цель системы – обеспечить съем и передачу информации из разнообразного оборудования в подсистему учета и принятия решений зернового терминала. Сеть объединяет и обеспечивает съем данных и управление следующим оборудованием: весовые автомобильная – ***Лахта СВ-60000А*** *(Петровес, Россия)* и железнодорожная – ***Лахта СВ-100000А***; разгрузки автомобильные и железнодорожная; отпускные морские весы – ***MWET*** *(Buhler, Германия)*; лабораторное оборудование для анализа качества зерна – высокоточные весы ***САРТОСМ ВР2100*** *(Сарториус, Россия)*, ядерно-магнитный резонатор – ***Minispec*** *(Bruker, Германия)*. Большая часть оборудования использует интерфейс RS-232 для управления и передачи данных. Каждая единица оборудования оснащена двумя CAN узлами: шлюзом RS232/CAN и модулем доступа, обеспечивающим авторизацию оператора с помощью бесконтактной ReadOnly смарт-карты (фирма EM-MARIN) и учет транспорта с зерном с помощью «таблетки» iButton (фирма DALLAS Semiconductor).

Таблица 2. Ресурсы, используемые модулями системы

	Модуль обмена (RS/CAN)	Модуль доступа	Проходная	Разгрузка
Микроконтроллер – SAB Siemens	C505CA	C505C	C505C	C505C
XRAM	256	256	256	256
CODE:				
Драйвер CAN	0.7 K	0.7 K	0.7 K	0.7 K
ОС РВ	0.5 K	0.5 K	0.5 K	0.5 K
CAL	1.5 K	2.5 K	2.5 K	2.5 K
Приложение	3.8 K	8.3 K	12.3 K	5.8 K
Все ПО	6.5 K	12 K	16 K	9.5 K

При разработке прикладного программного обеспечения был использован пакет DµCAL. Данный пакет разработан на языке ANSI C. При реализации модулей пакета использовались базовые программные средства ОС РВ DµOS-51 2.0 /8/ и драйвер CAN канального уровня /9/. Благодаря тому, что программы пакета написаны на языке С, он в процессе выполнения работы при незначительных затратах был портирован для ОС РВ RTX-51 Tiny (Keil Software), MS DOS и Windows 9x/NT. Основная проблема состояла в переносе драйвера CAN, так как использовались существенно разные ОС и различные CAN контроллеры.

На написание прикладного ПО было затрачено около 2 человекомесяцев. Реализация подобного распределенного приложения без использования CAL потребовала бы существенно больших временных затрат при большом количестве ошибок.

Литература

1. Jens Uphoff, Theory and Practical Experience with CAL-based Industrial Control, Proc. of the First Int. CAN Conf. (iCC'94), CiA, Erlangen, Germany

2. Wolfhard Lawrenz: «Vernetzte Systeme im Automobil dez Zukunft - Modellbildung», Hohere Lagen, Werkzeuge, forderkennzeichen: TV8933 Prometheus Phase II, TIB Hannover, September 1993

3. Третьяков С.А., «CAN на пороге нового столетия», МКА, № 2, с.54-65, 1999

4. CiA/DS201/DS207 CAL Specification, Version 1.1, February 1996

5. Wolfhard Lawrenz. CAN System Engineering., Springer, 1997

6. Erich-Jurgen Heins, A Real-time PC with iRMX for Windows controls Medical System Components directly via a CAN Network, Proc. of the First Int. CAN Conf. (iCC'94), CiA, Erlangen, Germany

7. CAN Specification. Version 2.0. 1991, Robert Bosch GmbH

8. ОС DµOS-51, v.2.0. Руководство программиста. DATAMICRO, Таганрог, октябрь 1998.

9. Драйвер CAN-контроллера, v.2.0. Руководство программиста. DATAMICRO, Таганрог, октябрь 1998.

10. Пакет CAL Slave, v.1.01. Руководство программиста. DATAMICRO, Таганрог, декабрь 1998.

Статья была впервые опубликована в «МКА – Мир Компьютерной Автоматизации» № 4, 1999.