Модульная шина сообщений

Темиргалеев Е. Э.
(Т.Е.Э. 9 ноя 2013 г. 19:07:11)

1. Постановка задачи

Исходная идея модульной шины, о которой идёт речь в данном сообщении, была представлена Губановым С. Ю. [1, сообщение p2457 [http://forum.oberoncore.ru/viewtopic.php?p=2457#p2457]] по аналогии с шиной «передачи сообщений между объектами». Передача сообщений объектам подразумевается согласно определению «A call *V.handle(V, M)* can therefore be interpreted as the sending of a message *M* to be handled individually by the method of the receiving viewer *V*.» [2, 3.1.1]. Ермаков И. Е. реализовал уточнённую в обсуждении [1, сообщение p2507 [http://forum.oberoncore.ru/viewtopic.php?p=2507#p2507]] идею, внеся дополнительные средства (компонент *AbfBus* [3]). Автор статьи согласно наблюдений о практическом применении сократил [1, сообщения с p70207[http://forum.oberoncore.ru/viewtopic.php?p=70207#p70207] по p70223[http://forum.oberoncore.ru/viewtopic.php?p=70223#p70223]] *AbfBus* до исходной идеи, суть исключив дополнения, результатом чего стал компонент *omcBus* (Приложение 1):

```
DEFINITION omcBus;
   CONST handlerMissed = -2; modNotFound = -1;
   TYPE HandleBusMsg = PROCEDURE (VAR msg: ANYREC);

   PROCEDURE Broadcast (VAR msg: ANYREC;
      OUT res: INTEGER);
   PROCEDURE BroadcastTo (IN sub: ARRAY OF CHAR; VAR msg: ANYREC;
      OUT res: INTEGER);
   PROCEDURE SendTo (IN module: ARRAY OF CHAR; VAR msg: ANYREC;
      forceLoad: BOOLEAN; OUT res: INTEGER);
END omcBus.
```

Цель работы — получить компонент для замены *omcBus*, выражающий концепцию шины более точно, сделав, по-возможности, его интерфейс более общим.

2. Анализ *omcBus*

2.1 Абоненты шины и широковещание. Основное назначение шины — широковещание. Применение широковещания в схеме взаимодействия уменьшает зависимости между

абонентами. Например: «Notice that broadcasting messages by a *model* (document) to the entirety of its potential *views* (viewers) is an interesting implementation of the famous MVC (*model-view-controller*) pattern that dispenses models from “knowing” (registering) their views.» [2, 4.2]. Более слабые зависимости закладывают потенциал возможного будущего развития.

Учитывая, что абоненты рассчитаны на приём произвольных сообщений, а обрабатывают известные им, широковещание может быть единственной операцией. Прочие, адресные варианты отправки суть оптимизация. Для модульшой шины *omcBus*, подключёнными абонентами которой являются загруженные модули, это хорошо видно в коде процедур (Приложение 1):

- широковещание (*Broadcast*): полный проход по списку модулей;
- широковещание подсистеме (*BroadcastTo*): полный проход по списку модулей с фильтрацией;
- адресная отправка (*SendTo*, *forceLoad* = FALSE): линейный поиск в списке модуля с данным именем.

Там же видно, что из этой схемы выбивается адресная отправка с требованием загрузки модуля (*SendTo*, *forceLoad* = TRUE). Суть отправка сообщения с подключением абонента. Отсюда (не вписывание в общую схему и комбинирование двух основных операций шины) делаем вывод о возможности исключить такую операцию.

2.2 Соединение с шиной; подключение и отключение. Модуль-абонент шины — экспортирующий процедуру-обработчик *HandleBusMsg* типа PROCEDURE (VAR *msg*: ANYREC). Подключение и отключение происходит неявно за счёт процессов, приводящих к загрузке и выгрузке модуля. Процедура-обработчик олицетворяет собой соединение с шиной.

Можно ввести (в данной модели шины) несколько шин за счёт добавления процедур обработчиков *HandleBusMsg_i*. Для сохранения универсального интерфейса это потребует явно выделить соединение с шиной. В самом общем случае соединение должно содержать имя процедуры-обработчика или, другими словами, имя шины.

Наличие соединения позволяет оптимизировать схему хранением адреса процедуры (в объекте, представляющем соединение). Это потребует от модулей-абонентов уведомления модуля шины о своей загрузке/выгрузке что суть операции подключения/отключения. Развивая мысль, приходим к явному введению этих операций с передачей адреса процедуры-обработчика модулем-абонентом:

- снимается требование получать адрес процедуры-обработчика посредством модуля *Meta* при каждом вызове — оптимизация;
- снимается требование экспорта процедуры-обработчика, т. е. выставления во внешний интерфейс внутренних элементов;
- момент подключения/отключения отвязывается от загрузки/выгрузки модуля — схема становится более гибкой;
- список абонентов отвязывается от списка модулей — хотя и усложнение реализации шины, но оптимизирующее, поскольку список модулей не совпадает со списком подключённых к шине модулей;
- имя шины отвязывается от имени процедуры-обработчика — схема становится более гибкой для реализации, вводя свободу выбора способа идентификации шин.

2.3 Применения

Из 19-ти рассмотренных случаев применения *omcBus* приблизительно 2/3 относятся к широковещанию и 1/3 к адресной отправки модулю. При этом около половины случаев адресной отправки составляют случаи "с требованием загрузки модуля", т. е. применяющие *omcBus.SendTo* как "удобную фичу" вместо прямой реализации вызова через *Meta*.

3. *PrivBus* — модульные шины сообщений

3.1 Интрефейс

В общем случае абоненты шины могут как отправлять, так и получать сообщения. В случае модульной шины (применение *omcBus*) выявился особый класс абонентов, которые только отправляют сообщения. Соответственно, параметры "имя абонента" и "обработчик сообщений" для них не имеют смысла. Для учёта этой особенности была введена классификация соединений на активные (отправляют и получают сообщения) и пассивные (только отправляют).

- параметры соединения указываются в момент подключения;

```
PROCEDURE Connect (VAR c: Connection; IN bus, abonent: Name; Handler: MsgHandler);
PROCEDURE Disconnect (VAR c: Connection);
PROCEDURE Connected (IN c: Connection): BOOLEAN;
```

- после подключения сообщения можно отправлять;

```
PROCEDURE Send (VAR c: Connection; IN receiver: Name; VAR msg: ANYREC);
PROCEDURE Broadcast (VAR c: Connection; VAR msg: ANYREC);
```

- активное соединение (*Hander* # NIL) при переходе в состояние "включено" начинает принимать отправляемые сообщения;

```
PROCEDURE Enable (VAR c: Connection);
PROCEDURE Disable (VAR c: Connection);
PROCEDURE Enabled (IN c: Connection): BOOLEAN;
```

Процедура *Send* не возвращает значение высказывания "абонент не подключен" по аналогии с *omcBus.SendTo* (выходной параметр *res*). Эта информация не имеет смысла, поскольку является частным случаем "абонент не обработал сообщение", который может быть выявлен только применением особенностей протокола взаимодействующих сторон. Например:

```
msg.n := -1;    (* недопустимое значение ответа *)
PrivBus.Send(c, receiver, msg);
IF msg.n = -1 THEN …    (* абонент receiver не обработал сообщение *)
```

Возможности запроса параметров соединения и состояния "активно" не введены, т. к. их необходимость не очевидна.

```
PROCEDURE GetConfig (IN c: Connection; OUT abonent: Name; OUT Handler: MsgHandler);
PROCEDURE Active (IN c: Connection): BOOLEAN;
```

Схема взаимодействия определяется не на уровне модулей, и эти параметры соединения должны быть доступны на этапе реализации модуля.

3.2 Применение вместо *omcBus*

Ниже приведены различия в исходных текстах модулей при переходе с *omcBus* на *PrivBus*. Так как последний допускает несколько шин, именем общей шины (аналога *omcBus*) выбрана пустая цепочка.

Только отправка сообщений (пассивное соединение *PrivBus*):

```
MODULE Xxx;
   IMPORT …, omcBus;
…
   VAR
      …

…
   PROCEDURE SendMsgs;
      VAR    …res: INTEGER;
   BEGIN
      omcBus.SendTo("A", m1, FALSE, res);
      omcBus.Broadcast(m2, res)
   END SendMsgs;
…

END Xxx.
```

```
MODULE Yyy;
   IMPORT …, MBus := PrivBus;
…
   VAR
      …
      mbusConn: MBus.Connection;
…
   PROCEDURE SendMsgs;
      VAR    …
   BEGIN
      MBus.Send(mbusConn, "A", m1);
      MBus.Broadcast(mbusConn, m)
   END SendMsgs;
…
BEGIN
   MBus.Connect(mbusConn, "", "", NIL);
   …
CLOSE
   …
   MBus.Disconnect(mbusConn)
END Yyy.
```

Приём сообщений (активное соединение *PrivBus*):

```
MODULE Xxx;

   IMPORT
      …;
…
   VAR …
…
   PROCEDURE HandleBusMsg* …
…

END Xxx.
```

```
MODULE Yyy;

   IMPORT
      …, MBus := PrivBus;
…
   VAR …; mbusConn: MBus.Connection;
…
   PROCEDURE HandleBusMsg …
…

BEGIN
   MBus.Connect(mbusConn, "", "", HandleBusMsg);
   MBus.Enable(mbusConn);
   …
CLOSE
   …
   MBus.Disable(mbusConn);
   MBus.Disconnect(mbusConn)
END Yyy.
```

3.3 Оценка времени вызова процедуры

1) Прямой вызов процедуры из другого модуля:
   Priv_devTimer.Proc(msg)

```
PROCEDURE Proc* (VAR msg: ANYREC);
BEGIN
END Proc;
```

2) Вызов процедуры-обработчика через *PrivBus*:
   PrivBus.Send(mbusConn, "PrivTimer", msg)

```
PROCEDURE BusHandler (VAR msg: ANYREC);
BEGIN
END BusHandler;
```

3) Вызов процедуры обработчика через *omcBus*:
   omcBus.SendTo("PrivTimer", msg, FALSE, res)

```
PROCEDURE HandleBusMsg* (VAR msg: ANYREC);
BEGIN
```

END HandleBusMsg;

В идеальных условиях — искомый абонент-получатель находится первым:
- *PrivBus* — один активный модуль на шине;
- *omcBus* — модуль-абонент *PrivTimer* только что загружен (стоит первым в списке *Kernel.modList* и будет первым выдан сканером *Meta* (BlackBox 1.6 [http://oberoncore.ru/projects/blackbox])),
получаем следующие отличия по времени:
- (2) выполняется приблизительно в 10 раз медленнее чем (1);
- (3) выполняется приблизительно в 100 раз медленнее чем (1).

Время поиска абонента для *PrivBus* и *omcBus* линейно зависит от числа абонентов. При этом для *omcBus* ситуация ухудшается тем, что:
- просматривается список всех модулей, а не только тех, в которых есть обработчик;
- время поиска обработчика *HandleBusMsg* в символьной информации зависит от числа экспортированных модулем процедур.

4. Список источников

1. Конференция "OberonCore". Тема t258[http://forum.oberoncore.ru/viewtopic.php?t=258] "Транспортная шина межмодульной передачи сообщений". (дата обращения: 09.11.2013)
2. Wirth N., Gutknecht J. Project Oberon. The Design of an Operating System and Compiler [http://oberoncore.ru/library/wirth_gutknecht_project_oberon_the_design_of_an_operating_system_and_compiler].
3. Подсистема Abf[http://oberoncore.ru/bbcc/subs/abf/].
4. Исходник *omcBus* [https://bitbucket.org/oberoncore/omc/src/1fca69ae42e2df201b5e7b5f32364e8cee184cc8/omc/Mod/Bus.odc?at=default]. // Репозиторий https://bitbucket.org/oberoncore/omc, изменение 1fca69a.

Приложение 1. Исходник *omcBus*

Копия [4] с поправками для лучшего восприятия (вырезаны штамп, оглавление; переставлены местами процедуры)⇨

```
MODULE omcBus;   (** <вырезан штамп 31 янв 2013 (0034)>   правки ⇨
- 20120131, ТЕЭ, Для пущего упрощения отпилена (вместе с зависимостью от Kernel и SYSTEM) подписка на сообщения: Subscribe, Unsubscribe, Send (можно сделать отдельным модулем)
- 20120131, ТЕЭ, Изменён интерфейс Subscribe/UnSubscribe➔➔
- 20120131, ТЕЭ, ➔поправки Broadcast и BroadcastTo➔
- 20120130, ТЕЭ, IsCap поправлена с русского на Latin-1. Наведение красот (CompareSub, SendMsg переписаны, субъективные оформительские мелочи)
- 20120130, ТЕЭ, Абстрактный Message заменён на ещё более абстрактный ANYREC.
- 20120130, ТЕЭ, Клонирование AbfBus->ombBus. Отпилена часть "Asynchronic Message Bus"
⇦
**)
(*
  (C) 2006, 2007 ЕИЕ
*)

  IMPORT Meta;

  CONST
    modNotFound* = -1;
    handlerMissed* = -2;
```

```
  TYPE HandleBusMsg* = PROCEDURE (VAR msg: ANYREC);

  VAR recursive: INTEGER; (* Counter of recursive calls to omcBus from HandleBusMsg *)


  PROCEDURE SendMsg (mod: Meta.Item; VAR msg: ANYREC; OUT ok: BOOLEAN);
    VAR
      item: Meta.Item;
      val: RECORD (Meta.Value)
        handler: HandleBusMsg
      END;
      done: BOOLEAN;
  BEGIN
    ASSERT(mod.Valid() & (mod.obj = Meta.modObj), 20);
    INC(recursive);
    mod.Lookup("HandleBusMsg", item);
    done := item.Valid() & (item.obj = Meta.procObj);
    IF done THEN item.GetVal(val, done) END;
    IF done THEN val.handler(msg) END;
    DEC(recursive);
    ok := done
  END SendMsg;


  PROCEDURE SendTo* (IN module: ARRAY OF CHAR; VAR msg: ANYREC; forceLoad: BOOLEAN;
OUT res: INTEGER);
    VAR item: Meta.Item; ok: BOOLEAN; sc: Meta.Scanner; name: Meta.Name;
  BEGIN
    ASSERT(module # "", 20);
    IF forceLoad THEN
      Meta.Lookup(module, item)
    ELSE
      sc.ConnectToMods;
      sc.Scan;
      sc.GetObjName(name);
      WHILE ~sc.eos & (name # module) DO
        sc.Scan;
        IF ~sc.eos THEN
          sc.GetObjName(name)
        END
      END;
      item := sc.this
    END;
    IF item.Valid() & (item.obj = Meta.modObj) THEN
      SendMsg(item, msg, ok);
      IF ok THEN
        res := 0
      ELSE
        res := handlerMissed
      END
    ELSE
      res := modNotFound
    END
  END SendTo;

  PROCEDURE IsCap (ch: CHAR): BOOLEAN;
  BEGIN
    RETURN ((ch >= "A") & (ch <= "Z") OR (ch >= "À") & (ch # "×") & (ch <= "Þ"))
  END IsCap;
```

```
  PROCEDURE CompareSub (IN modName, sub: ARRAY OF CHAR): BOOLEAN;
    VAR i, len: INTEGER; result: BOOLEAN;
  BEGIN
    len := LEN(sub$);
    result := len < LEN(modName$);
    IF result THEN
      i := 0;
      WHILE (i < len) & (sub[i] = modName[i]) DO
        INC(i)
      END;
      result := (i = len) & IsCap(modName[i])
    END;
    RETURN result
  END CompareSub;

  PROCEDURE BroadcastTo* (IN sub: ARRAY OF CHAR; VAR msg: ANYREC; OUT res: INTEGER);
    VAR sc: Meta.Scanner; modName: Meta.Name; ok: BOOLEAN;
  BEGIN
    sc.ConnectToMods;
    res := 0;
    sc.Scan;
    WHILE ~sc.eos DO
      IF sc.this.Valid() THEN
        sc.GetObjName(modName);
        IF CompareSub(modName, sub) THEN
          SendMsg(sc.this, msg, ok); IF ok THEN INC(res) END
        END
      END;
      sc.Scan
    END
  END BroadcastTo;

  PROCEDURE Broadcast* (VAR msg: ANYREC; OUT res: INTEGER);
    VAR sc: Meta.Scanner; ok: BOOLEAN;
  BEGIN
    sc.ConnectToMods;
    res := 0;
    sc.Scan;
    WHILE ~sc.eos DO
      IF sc.this.Valid() THEN
        SendMsg(sc.this, msg, ok); IF ok THEN INC(res) END
      END;
      sc.Scan
    END
  END Broadcast;

END omcBus.
⇦
```

Приложение 2. Исходник *PrivTimer*

```
MODULE PrivTimer;   (* © Темиргалеев Е. Э.,  2013 *)

  IMPORT Log, Services, omcBus, MBus := PrivBus, T := Priv_devTimer;

  VAR mbusConn: MBus.Connection;
  VAR msg: RECORD END;
  VAR
    nT,   (* число замеров  времени  *)
    nC: INTEGER;   (* число вызовов  для одного  замера  *)
```

```
  PROCEDURE BusHandler (VAR msg: ANYREC);
  BEGIN
  END BusHandler;

  PROCEDURE HandleBusMsg* (VAR msg: ANYREC);
  BEGIN
  END HandleBusMsg;

  PROCEDURE M1 (OUT time: LONGINT);
    VAR   i: INTEGER;
  BEGIN
    time := Services.Ticks();
    i := 0; WHILE i < nC DO T.Proc(msg); INC(i) END;
    time := Services.Ticks() - time
  END M1;

  PROCEDURE M2 (OUT time: LONGINT);
    VAR   i: INTEGER;
  BEGIN
    time := Services.Ticks();
    i := 0; WHILE i < nC DO MBus.Send(mbusConn, "PrivTimer", msg); INC(i) END;
    time := Services.Ticks() - time
  END M2;

  PROCEDURE M3 (OUT time: LONGINT);
    VAR   i: INTEGER; res: INTEGER;
  BEGIN
    time := Services.Ticks();
    i := 0; WHILE i < nC DO omcBus.SendTo("PrivTimer", msg, FALSE, res); INC(i) END;
    time := Services.Ticks() - time
  END M3;

  PROCEDURE Measure (P: PROCEDURE (OUT time: LONGINT); OUT mtime: REAL);
    VAR   i: INTEGER; time: LONGINT;
  BEGIN
    time := Services.Ticks(); WHILE Services.Ticks() = time DO END;
    mtime := 0;
    i := 0;
    WHILE i < nT DO
      P(time); mtime := mtime + time;
      INC(i)
    END;
    mtime := mtime / nT
  END Measure;

  PROCEDURE Run* (nt, nc: INTEGER);
    VAR   mtime1, mtime2, mtime3: REAL;
  BEGIN
    ASSERT(0 < nt, 20); ASSERT(0 < nc, 21);
    nT := nt; nC := nc;
    Measure(M1, mtime1); Log.Real(mtime1 / nC); Log.String(" мс");
    Log.Ln;
    Measure(M2, mtime2); Log.Real(mtime2 / nC); Log.String(" мс");
    Log.Tab; Log.Real(mtime2 / mtime1); Log.Ln;
    Measure(M3, mtime3); Log.Real(mtime3 / nC); Log.String(" мс");
    Log.Tab; Log.Real(mtime3 / mtime1); Log.Ln;
    Log.Ln
  END Run;

BEGIN
```

```
  MBus.Connect(mbusConn, "PrivTimer", "PrivTimer", BusHandler);
  MBus.Enable(mbusConn)
CLOSE
  MBus.Disable(mbusConn);
  MBus.Disconnect(mbusConn)
END PrivTimer.
```

❗"PrivTimer.Run(100, 1000000)"

Приложение 2.1 Исходник *Priv_devTimer*

```
MODULE Priv_devTimer;   (* © Темиргалеев Е. Э.,  2013 *)

  PROCEDURE Proc* (VAR msg: ANYREC);
  BEGIN
  END Proc;

END Priv_devTimer.
```

Приложение 3. Исходник *PrivBus*

```
MODULE PrivBus;   (* © Темиргалеев Е. Э.,  2013 *)

  TYPE
    MsgHandler* = PROCEDURE (VAR msg: ANYREC);
    Name* = ARRAY 256 OF CHAR;

    Connection* = RECORD
      c: Item
    END;

    Item = POINTER TO RECORD
      name: Name;
      bus, conn, cc: Item;
      Handler: MsgHandler
    END;

  VAR sentinel, buses, free: Item;


  PROCEDURE Connected* (IN c: Connection): BOOLEAN;
  BEGIN
    RETURN c.c # NIL
  END Connected;

  PROCEDURE Enabled* (IN c: Connection): BOOLEAN;
  BEGIN
    RETURN (c.c.Handler # NIL) & (c.c.cc # NIL)
  END Enabled;

  PROCEDURE Connect* (VAR c: Connection; IN bus, abonent: Name; Handler: MsgHandler);
    VAR   i: Item; bp, bi, cp, ci, new: Item;
  BEGIN
    ASSERT(c.c = NIL, 20);
    ASSERT((Handler # NIL) OR (abonent = ""), 21);
    bp := buses; bi := bp.bus;
    WHILE ~(bus$ <= bi.name$) DO bp := bi; bi := bi.bus END;
    IF bus$ = bi.name$ THEN
      i := bi
    ELSE
      IF free # NIL THEN
```

```
      new := free; free := new.conn; new.conn := NIL
    ELSE
      NEW(new)
    END;
    new.name := bus$;
    new.bus := bi; bp.bus := new;
    new.conn := sentinel;
    new.cc := sentinel; new.Handler := NIL;
    i := new
  END;
  ASSERT(i # buses, 22);
  IF Handler = NIL THEN
    c.c := i
  ELSE
    cp := i; ci := cp.conn;
    WHILE ~(abonent$ <= ci.name) DO cp := ci; ci := ci.conn END;
    IF abonent$ = ci.name$ THEN
      ASSERT(ci # sentinel, 23);
      HALT(24)
    ELSE
      IF free # NIL THEN
        new := free; free := new.conn; new.conn := NIL
      ELSE
        NEW(new)
      END;
      new.name := abonent$;
      cp.conn := new; new.conn := ci;
      new.bus := i; new.cc := NIL; new.Handler := Handler;
      c.c := new
    END
  END
END Connect;

PROCEDURE Disconnect* (VAR c: Connection);
  VAR  i: Item; cp: Item;
BEGIN
  i := c.c;
  IF i = NIL THEN
    (* c.c = NIL *)
  ELSIF i.Handler = NIL THEN
    c.c := NIL
  ELSE (* i.Handler # NIL *)
    ASSERT(i.cc = NIL, 20);
    cp := i.bus; WHILE cp.conn # i DO cp := cp.conn END;
    cp.conn := i.conn; i.conn := NIL;
    i.name := ""; i.bus := NIL; (* i.cc = NIL *) i.Handler := NIL;
    c.c := NIL;
    i.conn := free; free := i
  END
END Disconnect;

PROCEDURE Enable* (VAR c: Connection);
  VAR  i: Item; cp, ci: Item;
BEGIN
  i := c.c;
  ASSERT(i.Handler # NIL, 20);
  IF i.cc = NIL THEN
    ci := i.conn; WHILE ~(ci = sentinel) & ~(ci.cc # NIL) DO ci := ci.conn END;
    cp := i.bus; WHILE ~(cp.cc = ci) DO cp := cp.cc END;
    cp.cc := i; i.cc := ci
```

```
    END
  END Enable;

  PROCEDURE Disable* (VAR c: Connection);
    VAR   i: Item; cp: Item;
  BEGIN
    i := c.c;
    ASSERT(i.Handler # NIL, 20);
    IF i.cc # NIL THEN
      cp := i.bus; WHILE ~(cp.cc = i) DO cp := cp.cc END;
      cp.cc := i.cc; i.cc := NIL
    END
  END Disable;

  PROCEDURE Send* (VAR c: Connection; IN receiver: Name; VAR msg: ANYREC);
    VAR   i: Item; ci: Item;
  BEGIN
    i := c.c; IF i.Handler # NIL THEN i := i.bus END;
    ci := i.cc; WHILE ~(ci = NIL) & ~(receiver$ = ci.name$) DO ci := ci.cc END;
    IF ~(ci = NIL) THEN ci.Handler(msg) END
  END Send;

  PROCEDURE Broadcast* (VAR c: Connection; VAR msg: ANYREC);
    VAR   i: Item; ci: Item;
  BEGIN
    i := c.c;
    IF (i.Handler # NIL) & (i.cc # NIL) THEN
      ci := i.bus.cc; WHILE ci # i DO ci.Handler(msg); ci := ci.cc END;
      ci := i.cc; WHILE ci # sentinel DO ci.Handler(msg); ci := ci.cc END
    ELSE
      ci := i.cc; WHILE ci # sentinel DO ci.Handler(msg); ci := ci.cc END
    END
  END Broadcast;

  PROCEDURE Init;
    VAR   i: INTEGER;
  BEGIN
    NEW(sentinel);
    i := 0; WHILE i < LEN(Name) - 1 DO sentinel.name[i] := 0FFFFX; INC(i) END;
    sentinel.name[i] := 0X;
    sentinel.bus := NIL; sentinel.conn := NIL; sentinel.cc := NIL; sentinel.Handler := NIL;
    NEW(buses);
    i := 0; WHILE i < LEN(Name) - 1 DO buses.name[i] := 0FFFFX; INC(i) END;
    buses.name[i] := 0X;
    buses.bus := buses;
    buses.conn := NIL; buses.cc := NIL; buses.Handler := NIL;
    free := NIL
  END Init;

BEGIN
  Init
END PrivBus.
```

Приложение 3.1 Документация реализации *PrivBus*

TYPE **MsgHandler*** = PROCEDURE (VAR msg: ANYREC)
Процедура-обработчик сообщения шины.

TYPE **Name*** = ARRAY 256 OF CHAR
Имя шины, соединения (с конкретной шиной).

В качестве барьерного в упорядоченных по возрастанию списках используется значение *sentinelName* = (0FFFFX, …, 0FFFFX, 0X).

TYPE **Connection***
Соединение с шиной:
- активное — отправляет и принимает сообщения;
- пассивное — только отправляет.

Состояние "готовность к приёму сообщений" для активного — включено или выключено. (Пассивное всегда не готово или выключено.)

c: Item
Ссылка на внутреннее описание соединения:
- активное — на *Item* "соединение" (*c.Handler* # NIL);
- пассивное — на *Item* "шина" (*c.Handler* = NIL).

TYPE **Item** = POINTER TO RECORD
Описатель шины или соединения.

name: Name
(шина) Имя шины.
(соединение) Имя соединения.

bus: Item
(шина) Следующая шина в списке шин.
(соединение) Шина соединения.

conn: Item
(шина) Список соединений.
(соединение) Следующее соединение в списке соединений шины.

Список соединений упорядочен по возрастанию *name* (*x.name*$ < *x.next.name*$), с барьерным элементом *sentinel*, ацикличен (барьерный элемент последний).

cc: Item
(шина) Список *активных соединений в состоянии "включено"* (ACCB).
(соединение) Следующее ACCB в списке соединений шины.

Список ACCB является подсписком соединений — организация такая же (барьерный элемент последний). Т. о.:
(*c*(Item).*cc* # NIL) = "*c* — ACCB"

Handler: MsgHandler
(шина) Не используется (NIL).
(соединение) Обработчик сообщения.

VAR **sentinel**: Item
Барьерный элемент — один для всех списков — соединений и ACCB:

sentinel.name = sentinelName
sentinel.bus = NIL
sentinel.conn = NIL
sentinel.cc = NIL
sentinel.Handler = NIL

VAR **buses**: Item
Заголовок списка шин. Список упорядочен по возрастанию *name* (*x.name*$ < *x.next.name*$), цикличен, с барьерным элементом, в качестве которого используется заголовочный.

buses.name = sentinelName
buses.bus = {buses, список пуст; | первый элемент списка.
buses.conn = NIL
buses.cc = NIL
buses.Handler = NIL

Иллюстрации⇨
Приложение 3.1.1 Иллюстрации
⇦

VAR **free**: Item
Односвязный (поле связи *conn*) список свободных *Item* .

Освобождающиеся *Item* в *Disconnect* вместо выбрасывания в мусор ставятся в начало списка. *Connect* по необходимости забирает *Item* из начала списка, используя NEW только когда список пуст.

PROCEDURE **Connected*** (IN c: Connection): BOOLEAN
result = c.c # NIL = "соединение *c* подключено к шине"

PROCEDURE **Enabled*** (IN c: Connection): BOOLEAN
result = (c.c.Handler # NIL) & (c.c.cc # NIL) = "*c* — ACCB"

Предусловие
NIL dereference c.c # NIL = Connected(c)

PROCEDURE **Connect*** (VAR c: Connection; IN bus, abonent: Name; Handler: MsgHandler)
Подключение отключённого соединения *c* к шине *bus* как:
- активного (*Handler* # NIL) абонента *abonent*;
- пассивного (*Handler* = NIL) абонента (*abonent* = "").

Предусловие
20 c.c = NIL = ~Connected(c)
21 (Handler # NIL) OR (abonent = "")
22 bus # sentinelName
23 abonent # sentinelName
24 "пара (bus, abonent) уникальна"

Постусловие
i = c.c
i # NIL
i.Handler = Handler
Handler = NIL
  i — имеющийся или добавленный элемент списока шин *buses*
  i.name = bus
Handler # NIL
  i — добавленный элемент списка *i.bus.conn*

i.bus.name = bus
i.name = abonent

PROCEDURE **Disconnect*** (VAR c: Connection)
Отключение соединения *c*.

Предусловие
20 (c.c.Handler # NIL) -> (c.c.cc = NIL) = ~Enabled(c)

Постусловие
c.c = NIL = ~Connected(c)
c'.c.Handler # NIL
c'.c исключён из списка c'.bus.conn
free = c'.c

PROCEDURE **Enable*** (VAR c: Connection)
Включение соединения *c*.

Предусловие
NIL dereference c.c # NIL = Connected(c)
20 i.Handler # NIL = "*c* — активное"

Постусловие
Enabled(c)

PROCEDURE **Disable*** (VAR c: Connection)
Выключение соединения *c*.

Предусловие
NIL dereference c.c # NIL = Connected(c)
20 i.Handler # NIL = "*c* — активное"

Постусловие
~Enabled(c)

PROCEDURE **Send*** (VAR c: Connection; IN receiver: Name; VAR msg: ANYREC)
Отправка сообщения *msg* абоненту *receiver* шины через соединение *c*.

Предусловие
NIL dereference c.c # NIL = Connected(c)

PROCEDURE **Broadcast*** (VAR c: Connection; VAR msg: ANYREC)
Широковещание на шине сообщения *msg* через соединение *c*.
"Самому себе" сообщение не отправляется.

Предусловие
NIL dereference c.c # NIL = Connected(c)

PROCEDURE **Init**
Инициализация значений *sentinel* и *buses*.

Постусловие
sentinel # NIL
...
buses # NIL
...
free = NIL

Приложение 3.1.1 Иллюстрации