И. Е. Ермаков
Компонентный Паскаль/Блэкбокс в задачах АСУТП
Демонстрационный материал к докладу в День Оберона - 2017
Видео докладов: https://www.youtube.com/playlist?list=PLoKr-_Vv5yq7Hdxq1edBtyxvkTehP4t09

Две ветки подходов к разработке систем управления:
1) Как программная система. Язык программирования. 3GL (Ada). Или выход на 4GL. Но язык, средства абстракции, развитая семантика и формализация. Развито в бортовых системах. Сложные системы, подобные транспортной диспетчеризации (ДД, ЖД, авиадиспетчеризация, корабельные системы).
О ярко выраженном ЯВУ-завязанном подходе можно получить представление из некоторой литературы: литература по Modula и Ada, типа Янг "Языки программирования реального времени". Басс, Клеменс, Кацман "Архитектура ПО" (корабельные системы и продуктовая линейка в этой сфере). Соммервил, Фокс... Да что там, даже Грейди Буч в своих книгах по C++/UML разбирает архитектуры таких систем.
Роскосмос - НПО Решетнева, Модула-2 и сквозное инструментирование.

2) Промышленная автоматизация по схеме блочной интеграции, сбора сигналов и простой циклической логики.
Интегрированные среды редактирования проектов.
Хотите представить инструментарий и уровень абстракции? Представьте разработку системы на MS Access. Попытки "сделать удобно и визуально" при полной немощи формализации, обобщения. Остутствие "единого исходника".
Плюсы, бесспорно, тоже есть.

Наши проекты.
Маршрутное производство.... Редактор маршрутов персоналу и т.п. Уже не накопипастишь. Пример с подрядчиками-голландцами и нашими.
В отрасле есть и неплохой самописный софт (беларусы на .NET, Воронеж, Алтай).
Реалии "неидеального мира", который надо оперативно абстрагировать, регуляризовывать и развивать прямо "в поле".

Гибриды: введение CASE-инструментария в подход-1, хорошее инструментирование, декларативность.
Введение скриптовых языков в подход-2 (жуткого качества. VBA, JS, C-скрипт. Пример - ошибка с памятью под строку вешает наглухо Siemens WinCC).

В итоге разработчики вынуждены прорабатывать систему объектов, метаинформации времени проектирования и времени исполнения, и частично, насколько осиливают, пускать к этому скриптовой рантайм.

Пример со Шнайдером и Vijeo Citect (нет единого инструмента для создания АРМ оператора, который можно загнать и на сенсорную панель и на ПК, нужно делать дурную двойную работу).
Windows. OPC, DCOM. Flash, Silverlight, Adobe... (3D-тренд).

Ценник - сотни тысяч, миллион.

Норма - некие SCADA типа российской MasterSCADA (100-150 тыс. р., внятная модель, нормальная реализация). Даже некий аппарат обобщения, но всё равно это "недопрограммирование".

Пакеты же типа WinCC...
Сам социально-экономико-технологически-психологический феномен: как??
Кто и почему в этих копрорациях разрабатывает ЭТО?

Замкнутая петля: огромная гора "примочек", чтобы дать инженерам не программировать, а собирать ("из известно чего и палок").
Но и сами реализаторы этих пакетов не способны даже заложить нормальную архитектуру и дать надёжную реализацию.

Метаинформация. Объект должен быть языковой сущностью. И должен быть обрабатываем из языка (генерационный подход). Пример - с закономерностью адресации.
Теряем в конфигурабельности? Можно сделать "виртуальный модуль", порождаемый из БД.
Часть вещей - стандартным языковым ООП.

Про ценность самой модели рантайма (ср. с Java и др.).

Пример самых простых приёмов, если делать "с нуля" SCADA на КП/ББ.

1) Тег как объект (DurT, симуляция, достоверность, уникальная индентификация...)
Генерационная привязка тегов к адресации.

```
IF tg.on.Val() & (tg.on.DurT() > checkOnSigAfterStart) THEN
  IF ~tg.onSig.Val() & (tg.on.DurT() > tg.onSig.DurT()) THEN
    Tok.Incl(tg.troubles, Tags.pathErr)
  END
END
```

(Опять обратим внимание на troubles - массив токенов, а не что-нибудь...)

```
Base* = POINTER TO EXTENSIBLE RECORD
  Q-: Tags.BoolOut;
  Q_NO-, Q_K-: Tags.Bool;
END;

BaseEx* = POINTER TO EXTENSIBLE RECORD (Base)
  repair-: Tags.Bool;
END;

SL* = POINTER TO RECORD (BaseEx)
```

```
    mov-: Tags.Bool;
  END;

  VE* = POINTER TO RECORD (BaseEx)
    QS-: Tags.BoolOut;
    QS_NO-, QS_K-: Tags.Bool;
    QT-: Tags.BoolOut;
    QT_NO-, QT_K-: Tags.Bool;
    expl-: Tags.Bool
  END;
```

Генерационная привязка:

```
  PROCEDURE InitEV (VAR ev: EV; peOffs, peBit, movOffs, movBit, ulOffs,
ulBit: INTEGER);
  BEGIN
    InitTKF(ev, peOffs, peBit, movOffs, movBit);

    Inc(peOffs, peBit);
    ev.QS_NO := pe.Bit(peOffs, peBit);
    ev.QS_K := pa.Bit(peOffs, peBit);
    ev.QS := mko.BitOut(peOffs, peBit);

    Inc(peOffs, peBit);
    ev.QT_NO := pe.Bit(peOffs, peBit);
    ev.QT_K := pa.Bit(peOffs, peBit);
    ev.QT := mko.BitOut(peOffs, peBit);

    ev.UL := pe.Bit(ulOffs, ulBit);
    Inc(ulOffs, ulBit);
    ev.UR := pe.Bit(ulOffs, ulBit);
    Inc(ulOffs, ulBit);
    ev.DL := pe.Bit(ulOffs, ulBit);
    Inc(ulOffs, ulBit);
    ev.DR := pe.Bit(ulOffs, ulBit)

  END InitEV;

  InitEV(ev_2_000, 10, 4, 110, 5, 212, 0);
  ev_2_000.repair := pe.Bit(116, 4);
  ev_2_000.over := pe.Bit(211, 6);
  ev_2_000.expl := pe.Bit(111, 2);
  InitEV(ev_2_010, 10, 7, 110, 4, 212, 4);
  ev_2_010.repair := pe.Bit(116, 5);
  ev_2_010.over := pe.Bit(211, 7);
  ev_2_010.expl := pe.Bit(111, 3);
  InitEV(ev_20_000, 20, 2, 120, 2, 227, 0);
  ev_20_000.repair := pe.Bit(220, 2);
  ev_20_000.over := pe.Bit(226, 0);
  ev_20_000.expl := pe.Bit(128, 0);
```

```
    InitEV(ev_20_100, 20, 5, 120, 3, 227, 4);
    ev_20_100.repair := pe.Bit(220, 3);
    ev_20_100.over := pe.Bit(226, 1);
    ev_20_100.expl := pe.Bit(128, 1);
    InitEV(ev_27_000, 23, 5, 120, 7, 228, 0);
    ev_27_000.repair := pe.Bit(222, 1);
    ev_27_000.over := pe.Bit(226, 2);
    ev_27_000.expl := pe.Bit(128, 2);
    InitEV(ev_29_000, 24, 7, 121, 3, 228, 4);
    ev_29_000.repair := pe.Bit(222, 7);
    ev_29_000.over := pe.Bit(226, 3);
    ev_29_000.expl := pe.Bit(128, 3);
```

Скажете, минус - "прибито гвоздями", вместо конфигурационной базы?
Вспоминаем идею из доклада 1:

==========
Идея любого CASE на базе ББ: исходник модуля - как комплект бинарных файлов.
Таблица деклараций, ДРАКОН-схема или конечный автомат.
При компиляции исходник собирается "на лету" и подаётся на вход компилятору.
Для рантайма - сгенерённый модуль, плюс вся метаинформация нужного типа.
==========

2) Обобщение узлов типами.

3) Iterate через Meta.

```
    Meta.Lookup(tagsModName, mod);
    ASSERT(mod.Valid(), 100);
    KrlMeta.IteratePtrs(KrlMeta.noRec, mod, IterHandler);

  PROCEDURE IterHandler (tg: ANYPTR; VAR context: ANYREC);
    VAR it: Meta.Item;
  BEGIN
    WITH tg: Tags.V0 DO
      ....
    | tg: Tags.SwitchOn DO
      ....
    | tg: Tags.Gate DO
      ...
    | tg: Tags.Valve DO
      ...
    ELSE
    END
  END IterHandler;
```

Конечно же, инструменты (инспекторы значений, симуляцию и др.)

делать над Meta - "вперёд и с песней".

4) WITH. Можно и консолидировать или разнести.

5) Слоение простой системы: Tags, Exchange, Logic, Ui.

6) Интеракторы.
Контроллер GUI.

```
PROCEDURE (c: Controller) Get_- (v: UC.Control; VAR rec: ANYREC; nd: ANYPTR; VAR stat: ANYREC);
BEGIN
  IF nd = NIL THEN
RETURN
  END;

  WITH nd: Tags.Node DO
    WITH stat: UC.Status DO

      IF v.tag = StButtons.tag THEN (* Проверяем тип таким образом, по полю контрола tag, сравнивая с тегом модуля *)

        (* Состояние кнопок пуск-стопа *)
        WITH nd: Tags.SoftButton DO
          IF nd.val.Val() THEN
            stat.state[0] := StButtons.picStop;
            stat.color[0] := Colors.redButton
          ELSE
            stat.state[0] := StButtons.picStart;
            stat.color[0] := Colors.greenButton
          END
        ELSE
          IF IsInError(nd) OR ~IsReady(nd) THEN
            stat.state[0] := StButtons.picNone;
            stat.color[0] := Colors.greyDetail
          ELSIF IsOn(nd) THEN
            stat.state[0] := StButtons.picStop;
            stat.color[0] := Colors.redButton
          ELSE
            stat.state[0] := StButtons.picStart;
            stat.color[0] := Colors.greenButton
          END
        END

      ELSIF v.tag = Gates.tag THEN

        IF IsOn(nd) & IsOnReal(nd) THEN
          IF IsOnToOpen(nd(Tags.Gate)) THEN
            stat.state[0] := Gates.posOpened
          ELSIF IsOnToClose(nd(Tags.Gate)) THEN
```

```
            stat.state[0] := Gates.posClosed
          ELSE
            HALT(100)
          END;
          stat.color[0] := Colors.starting.color
        ELSE
          stat.state[0] := GatePos(nd(Tags.Gate));
          IF IsInError(nd) THEN
            stat.color[0] := Colors.error.color;
            stat.blitMode[0] := Colors.error.blitMode
          ELSIF ~IsReady(nd) OR (stat.state[0] = Gates.posUnknown) THEN
            stat.color[0] := Colors.error.color
          ELSE
            CASE stat.state[0] OF
            | Gates.posOpened:
              stat.color[0] := Colors.run
            | Gates.posClosed:
              stat.color[0] := Colors.greyDetail
            END
          END
        END
```

6) Предикаты о состояниях часто удобнее хранимых состояний. ASSERT для гарантии правильного вызова (разбиения пространства состояний).

```
  PROCEDURE IsOn (nd: Tags.Node): BOOLEAN;
    VAR on: BOOLEAN;
  BEGIN
    on := FALSE;
    WITH nd: Tags.SwitchOn DO
      on := nd.on.Val()
    | nd: Tags.Gate DO
      on := nd.close.Val() OR nd.open.Val()
    | nd: Tags.Valve DO
      on := nd.toPos1.Val() OR nd.toPos2.Val()
    ELSE
    END;
  RETURN on
  END IsOn;

  PROCEDURE IsOnReal (nd: Tags.Node): BOOLEAN;
    VAR real: BOOLEAN;
  BEGIN
    ASSERT(IsOn(nd), 20);
    real := TRUE;
    WITH nd: Tags.SwitchOn DO
      real := nd.onSig.Val()
    ELSE
    END;
  RETURN real
```

```
END IsOnReal;

PROCEDURE IsOnLocal (nd: Tags.Node): BOOLEAN;
  VAR on: BOOLEAN;
BEGIN
  on := FALSE;
  WITH nd: Tags.SwitchOn DO
    IF nd.onSig # NIL THEN
      on := ~nd.on.Val() & nd.onSig.Val()
    END
  ELSE
  END;
RETURN on
END IsOnLocal;

PROCEDURE On (nd: Tags.Node);
BEGIN
  WITH nd: Tags.SwitchOn DO
    nd.on.Set(TRUE)
  ELSE
  END
END On;

PROCEDURE Off (nd: Tags.Node);
BEGIN
  WITH nd: Tags.SwitchOn DO
    nd.on.Set(FALSE)
  ELSE
  END
END Off;

PROCEDURE IsStarting (nd: Tags.Node): BOOLEAN;
  VAR starting: BOOLEAN;
BEGIN
  ASSERT(IsOn(nd), 20);
  starting := FALSE;
  WITH nd: Tags.SwitchOn DO
    starting := ~nd.onSig.Val()
  ELSE
  END;
RETURN starting
END IsStarting;

PROCEDURE IsStopping (nd: Tags.Node): BOOLEAN;
BEGIN
  ASSERT(~IsOn(nd), 20);
RETURN FALSE
END IsStopping;
```