Сепарация кода от данных как основа программной архитектуры высоконадежных распределенных систем

Дагаев Дмитрий Викторович, Генеральный директор ООО «СКАДИ»,
Консультант проекта Информатика-21

О требованиях функциональной безопасности

- “Надежная программа – прежде всего, должна обеспечивать достаточно низкую вероятность отказа в процессе функционирования в реальном времени. Быстрое реагирование на искажения программ, данных или вычислительного процесса и восстановление работоспособности за время меньшее, чем порог между сбоем и отказом, обеспечивают высокую надежность программ.” В.В.Липаев
  - Отказоустойчивость и резервирование: от принципа единичного отказа к живучести при многих, известны системы до 13 уровней *элегантной деградации*;
  - Диверсификация или *принцип разнообразия*;
  - Сохранение в восстановление *состояний*;
  - Ремонтопригодность и возможность *горячей замены*;
  - Исключение взаимовлияния компонентов;
  - *Эргодичность* и уменьшение старения клеток;
  - *Ограничение сложности* ПО.

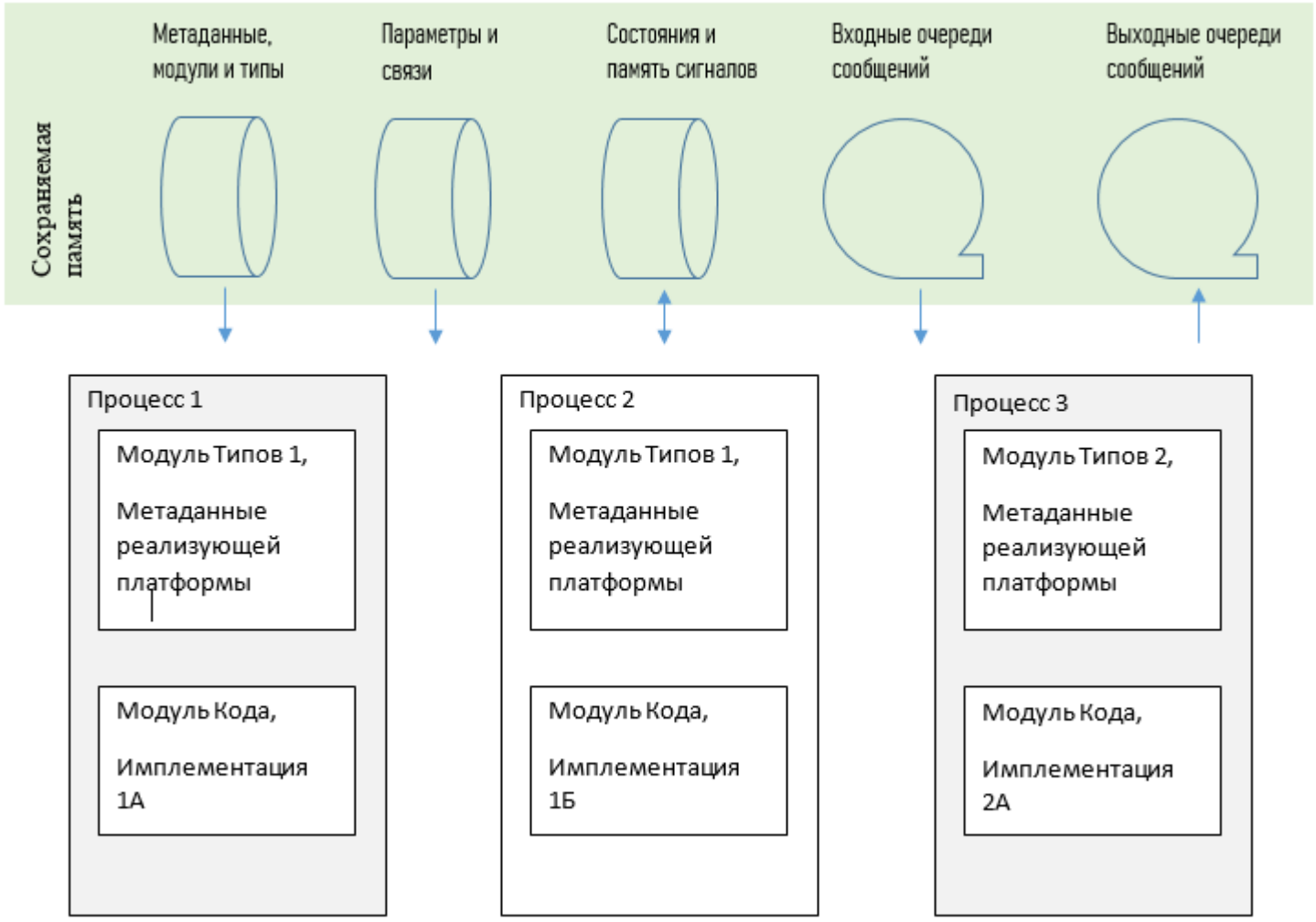
Увеличить наработку на отказовую ситуацию, уменьшить время восстановления

- Методы увеличения наработки на отказовую ситуацию:
  - Уменьшение сложности ПО;
  - Система семантических ограничений RESTRICT;
  - Повышение качества.
- В пределе – исключить программный код, оставить только данные.

- Методы уменьшения времени восстановления:
  - Самодиагностика;
  - Возможность горячей замены;
  - Восстановление состояний после сбоев, рестартов.
- В пределе – исключить данные, оставить только код.

Время жизни системы больше времени жизни ненадежных компонентов

Программирование, управляемое данными DOP



Принципы DOP Евгения Кузнецова (IBM) упоминаемого как автора, на которых реализован ряд дата-центричных технологий.

- Сепарация кода от данных;
- Предоставление данных и метаданных на основании прав доступа вместо инкапсуляции;
- Обработка данных на основе метаданных реализующей платформы;
- Скрытие подробностей кода.

Реализация средствами модульного программирования Оберон.

Сомнительность объединения кода и данных в ООП?

Hanspeter Mössenböck Oberon-2:

Type-bound procedures

CLOP Class-Oriented procedures

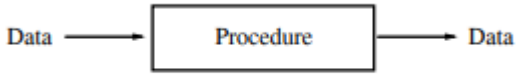


Fig. 1.1 Procedure-oriented thinking

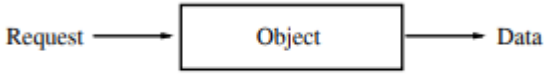


Fig. 1.2 Object-oriented thinking

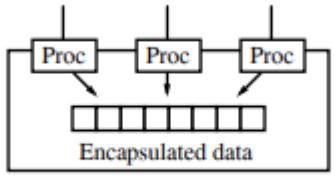
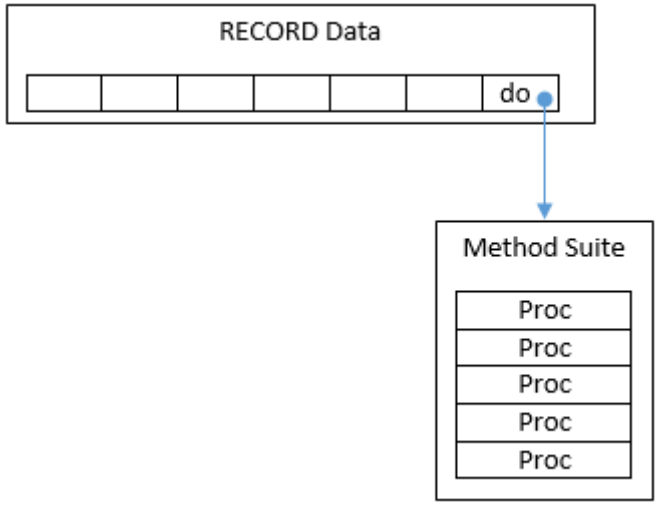


Fig. 1.3 An object with encapsulated data and a procedural interface

Niklaus Wirth Jürg Gutknecht Project Oberon:

Suite-bound procedures

Методы инсталлируются в объекте Method Suit, отделенном от основного Data



Двойная роль ресивера

«Отметим, что ресивер играет двойную роль: Во-первых, он передается как параметр метода и во-вторых, объект, заключенный в нем, определяет, какой метод нужно вызывать.»
H.Mössenböck .

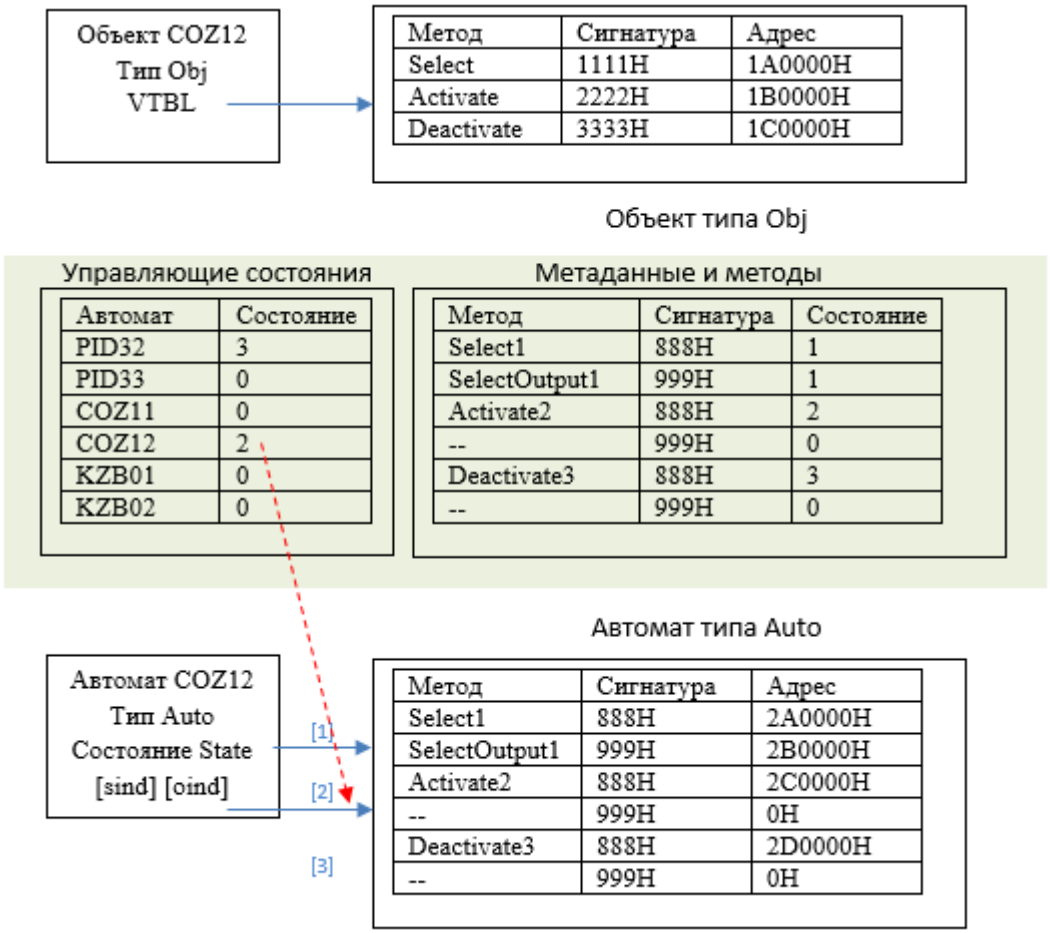
`obj.Write(data)` – означает

1. Вызов метода _Write(obj, data)
2. Выбор метода _Write := TYPE(obj).Procs[indexWrite]

`obj.do.Write(obj, data)` – означает

1. Вызов метода _Write(obj, data)
2. Выбор метода _Write := obj.do – метод отделен от объекта

Решение проблема ООП для автоматной модели



- Управляющие состояния сохраняются в общей памяти;
- Отделенные от данных коды процедур представляют собой массив от sind – состояния конечного автомата;
- Проблема: Один фактический параметр – массив ресиверов.

```
do := MetaData[obj.otype].do;
do[obj.sind].State(obj,
obj.sind);
```

Автомат с состояниями выходит за рамки ООП

- Для состояния 1 – один набор методов.

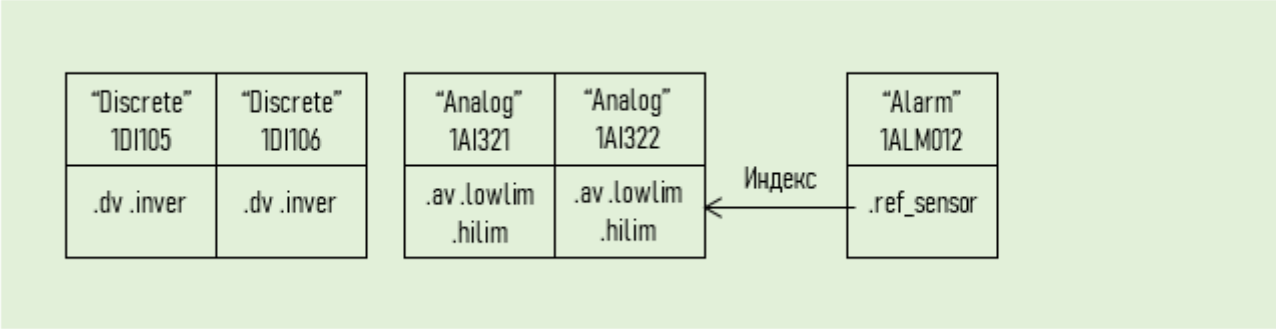
```
do[1].State := Select1;
do[1].Output := SelectOutput1;
```

- Для состояния 2 – другой набор методов.

```
do[2].State := Activate2;
do[2].Output := EMPTY;
```

При изменении номера состояния меняется тип объекта «состояние».

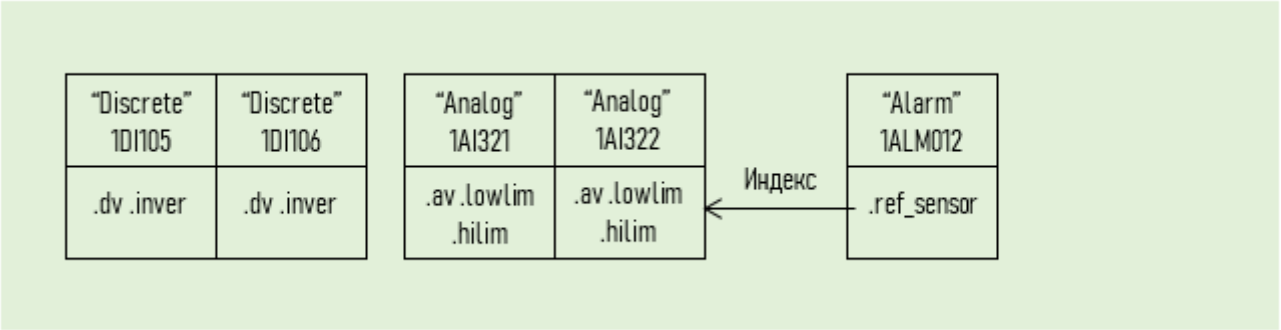
Обработка входных разнотиповых данных



Проблема: Один ресивер – набор фактических параметров

1. Создание копий объектов данных (массив интерфейсов)
   ```
   obj[n] := types[data[n].otype].New(data);
   ```
2. Вызов виртуального метода обработки
   ```
   obj[n].Process();
   ```

Обработка входных разнотиповых данных с сепарацией



1. Нахождение объекта с методами
   `obj := Metadata[data[n].otype].do;`
2. Вызов инсталлированного метода обработки
   `obj.Process(data[n]);`

Данные отделены от кода, данные являются персистентными (а код ненадежен), данные потенциально бесконечны (может быть СУБД – SELECT, FETCH, …)

Вырожденный случай – Код есть, данных нет

- Пример – реализация Log для встроенки (без использования NEW);
- Проблема: Один ресивер – нет фактических параметров;
- Вопрос о методах в Obernon-07.

```
Module Log;
TYPE Method = RECORD … END;
VAR hook-: Method;
PROCEDURE Install*; BEGIN
      hook.String := StringProc;
      hook.Int := IntProc;
      hook.Ln := LnProc;
END Install;
END Log;
Module Test;
BEGIN
      Log.hook.String(“Test”); Log.hook.Ln
END Test;
```

Утиные интерфейсы в классическом Обероне

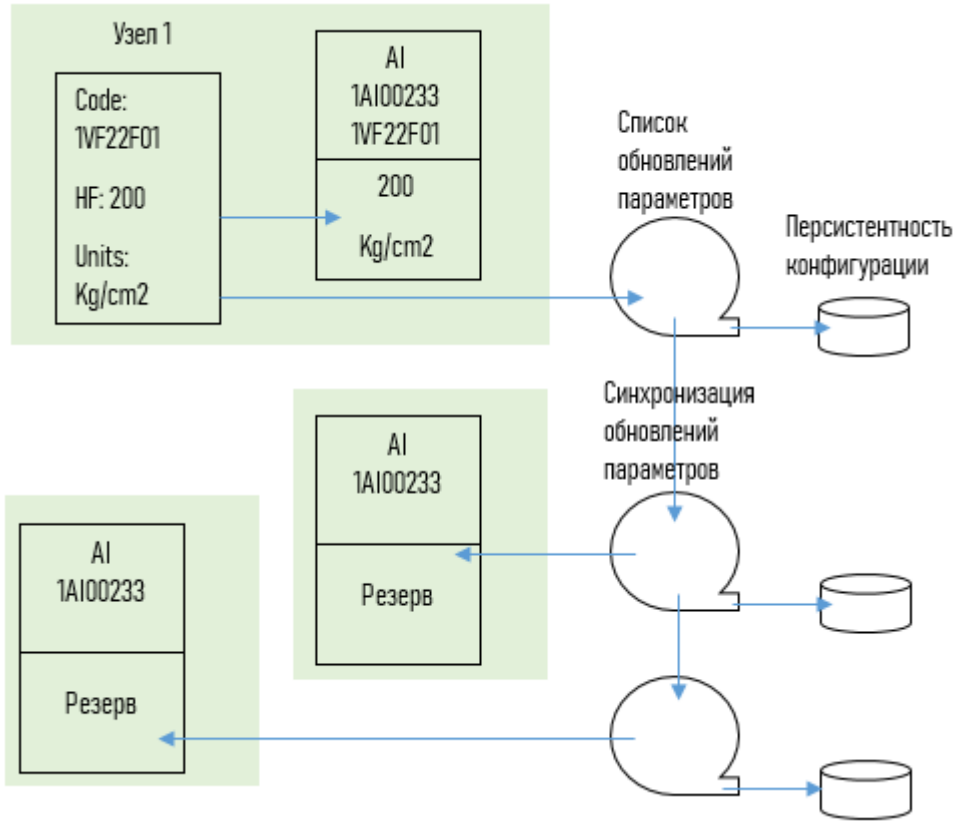
Проблема: ресивер Stringer в иерархии типов никак не относится к объектам IntObj, RealObj.

Интерфейс ресивера в классическом Обероне может реализуется методом инкапсуляции obj.s.

Вопрос только в удобном получении (типа obj(Stringer)).

```
Module Duck;

Stringer* = POINTER TO RECORD toString = PROCEDURE(obj: ANYREC) END;

TYPE IntObj = RECORD …; s: Stringer END;
RealObj = RECORD …; s: Stringer END;

PROCEDURE Print*(obj: ANYREC);

VAR s: Stringer;

BEGIN
      s := obj(Stringer); (* obj.s *)
      s.toString(obj);
END Print;

END Duck;
```

Возможность синхронизации изменений



Проблема: Вследствие инкапсуляции механизм изменений параметров объектов в распределенных системах становится проблематичным.

Мы проводим изменение конфигурации сборки в режиме онлайн.

Изменения параметров являются персистентными и сохраняют свои значения при возобновлении питания.

Реализована проверка целостности на узлах на протяжении всего времени жизни системы.

Данные, отделенные от кода, живут дольше, чем инкапсулированные

Некоторые выводы

- Сепарация данных от кода дает возможность разделения их жизненных циклов в контексте развития/обновления;
- Переход в Оброне-2 к типо-ориентированным процедурам несет за собой набор ограничений, несвойственный классическому Оберону;
- Инкапсуляция данных, предлагаемая ООП, не всегда является очевидным преимуществом при построении систем.

Дагаев Дмитрий Викторович dvdagaev@oberon.org

Вопросы по докладу …