ПИСКУНОВ А.Г., ИЛЮХИН В.Л.

ИСПОЛЬЗОВАНИЕ ЯЗЫКОВ ФОРМАЛЬНЫХ СПЕЦИФИКАЦИЙ ПРИ ПРОЕКТИРОВАНИИ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

The usage of formal specification languages for the design of RDBMS

We discuss an application of formal specification languages, particularly RSL (cm. [2],[3]), to the design of RDBMS. Small project GSAU has been taken as an example. There are two pairs of tables, linked by one-to-many relationships describe in the project. The first pair contains actual state of gas pumps and the second one supports transmission of control signals from the operator to the pump controller.

Авторы высказывают благодарность людям, внесшим свою лепту в данную работу: C.George, О.Головко, В.Бубнов, М.Николаев, N.Levental, W.Mutual, С.Пономарев, А.Павлов, М.Роньшин, В.Проценко, Д.Орлов.

В работе обсуждается использование языков формальных спецификаций, в частности, RSL (cm. [2],[3]) для проектирования реляционных баз данных на примере небольшого проекта GSAU (cm. [8]) - "Коммуникационная Компонента Системы Управления Заправочной Станцией". В проекте описываются две пары таблиц, попарно связанных отношением один- ко - многим. Одна пара содержит реальное состояние насосов заправочной станиции, другая служит для передачи управляющих сигналов от операторов к контроллеру насосов.

Обсуждение других языков формальных спецификаций (Z, VDM-SL, Larch) можно прочитать в книге [12] или в интернет-ресурсах [13].

ВВЕДЕНИЕ

После нескольких лет изучения литературы, касающейся вопросов

1. построения реляционных баз данных ([1]);
2. построения надежного программного обеспечения ([6]);
3. использования формальных методов для построения надежного программного обеспечения (RAISE: [2], [3]; VDM: [7]),

и, пообщавшись с некоторыми разработчиками и проектантами, удалось заметить, что программисты, которые работают в данных областях, как бы игнорируют друг друга. То есть те, кто использует формальные методы, ни чего не знают про реляционные базы данных, а те, кто занимется теорией и практикой реляционных базы данных, ведать не ведают о формальных подходах к построению программного обеспечения.

В документе делается попытка показать пути для использования языка формальных спецификаций RAISE для проектирования БД и наметить возможности по автоматической генерации SQL - скриптов и кода, написанного на процедурных языках (например C++) из формальных спецификаций языка RAISE. Таким образом, используя мощь формальной пошаговой разработки и жесткой проверки типов на ранних стадиях RAISE Development Method-а, можно было бы добиться существенного прогресса в таких нетривиальных вопросах, например, как согласованное измененение типов полей таблиц в SQL - скриптах языка определения данных и, соответствующих этим полям, переменных процедурного языка.

В связи с очевидной необьятностью данной задачи в общем (замахиваться на генерацию всех тонкостей, например, SQL от Oracle - на наш взгляд бесперспективная задача), было принято решения сконцентрироваться на подмножестве DML (язык манипулирования данными), как наиболее близкий к "чистому" SQL и оставить без рассмотрения всевозможные диалекты для создания хранимых процедур (PL/SQL, Transact SQL, Watcom SQL и так далее). В результате удалось

1. создать специальный тип отношение Relation, как отображение из натуральных чисел(поля первичного ключа) в неинтерпретируемую запись (собственно сама запись);
2. ввести функции (-члены класса), моделирующие все четыре операци (Insert, Update, Delete, Select с учетом организации курсоров);
3. ввести функции (-члены класса), моделирующие фразу where (where);
4. ввести функции (-члены класса), моделирующие проверку всевозможных ограничений целостности (check, unique);
5. промоделировать условия ссылочной целостности (referential integrity);
6. показать способ записывания триггеров;
7. показать какие поля структур должны иметь умолчательные значения и могут не вводится в БД, а какие обязательно должны вводится явно.

Со стороны клиентской части приложения (процедурный язык высокого уровня) была создана Библиотека Структур Обмена с базой данных ([4], [5]), основанная на идее использования для обмена (Select, Update, Insert) информацией не отдельных host- переменных, а специальных структур обмена, которые автоматически создаются из определения таблицы или обзора. Причем удалось добиться "существенной похожести" обьектов, наследующих свойства типа отношеня Relation и структур обмена, таким образом, что одна строчка языка формальных спецификаций транслируется в одну (ну почти одну) строчку процедурного языка. В данном случае - C++.

Все вышеизложенное позволяет считать что создан некоторый подход, позволяющий проектировать и разрабатывать приложения, основанные на реляционных базах данных таким формальным методом как RAISE Development Method. В качестве иллюстрации по использованию данного подхода приводится пример разработка протокола общения между контроллером насосов и клиентскими местами (операторами) заправочной станции.

ОБЩАЯ СХЕМА КОМПОНЕНТ GSAU И ИНТЕРФЕЙСОВ МЕЖДУ НИМИ

Согласно рекомендациям ([6]) определим компоненты из которых состоит система GSAU и проследим потоки данных между компонентами.

Список компонент системы GSAU:

1. исполнитель (agent) - человек, который непосредственно пользуется насосом и наливает горючее в бак автомобиля;
2. бак автомобиля, далее бачок;
3. шланг насоса (nozzle);
4. насос (pump); кнопка (позиция) выбора горючего, далее кнопка (position). С каждой кнопкой жестко связана некотороя пропорция в которой смешивается (или не смешивается) горючее из разных баков заправочной станции. ;
5. бак заправочной станции, далее бак (tank). На заправочной станции содержится по два бака горючего разного качества. В момент заправки горючее из двух баков смешивается в заданной пропорции;
6. контроллер - устройство передающиее информацию между насосами и компьютером;
7. com - порт (com port) - устройство, через которое контроллер подключается к компьютеру.
8. драйвер контроллера (driver) - программа, которая позволяет обменивается информацией контроллеру (через com- порт) и приложению, использующему контроллер. В нашем случае это \nm;
9. иммитатор драйвера контроллера - программа, позволяющая использовать GSAU без физического наличия драйверов контроллера, com- портов, контроллеров и насосов (в основном для целей отладки и демонстрации работы GSAU). Запускается на GSAU- сервере.
10. иммитатор контролера - программа, позволяющая использовать GSAU без физического наличия контроллеров и насосов (в основном для целей отладки и демонстрации работы GSAU). Требуется драйвер контроллера, com- порт, дополнительный компьютер (на котором будет запускаться имитатор контроллера) и нуль- модем для соединения com- портов на GSAU -сервере и дополнительном компьютере.
11. менеджер драйверов контроллера (manager) - программа, маскирующая от основной части GSAU особенности реализации и сам факт наличия того или иного контроллера.
12. наблюдатель за корректростью изменения состояния насоса (pump monitor), наблюдатель состояний.
13. база данных - не комментируется.
14. UDP/TCP отправитель и получатель - компоненты GSAU, для передачи информации через UDP/TCP протоколы соответственно.
15. приложение интерфейса оператора (UI) - основная компонента GSAU - клиента, позволяющая оператору GSAU наблюдать за работой насосов заправочной станции и управлять их работой.
16. оператор.

Драйвера, менеджер драйверов, наблюдатель состояния, TCP получатель, UDP отправитель и база данных физически располагаются на GSAU - сервере (Unix box). Наблюдатель ввода и приложение интерфейса пользователя, UDP получатель, TCP отправитель - физически располагаются на GSAU - клиенте (Windows box).

1. Описание взаимодействия GSAU- сервера и GSAU- клиента

GSAU- сервер и GSAU- клиент обмениваются информацией друг с другом в асинхронном режиме через две пары каналов. (Вообще говоря GSAU- клиентов может быть больше одного, но этот вопрос в данный момент нас не интересует и на схеме не отражен). В каждой из пар роль одного канала играет соответствующий протокол интернета, роль второго канала играет соответствующая таблица БД. Каждая из компонент может послать информацию в другую компоненту в любой момент времени. От сервера к клиенту (клиентам) информация посылается через протокол UDP (на схеме - проходит через канал UDP, таблицу State), так как все клиенты должны отображать все изменения, которые происходят на насосах станции. Таким образом получаем одного отправителя и много получателей информации. От клиента (клиентов) к серверу информация посылается через протокол TCP

(канал TCP и таблицу Command), так как имется в точности один отправитель и один получатель информации.

В каждой отдельной операции пересылки/требования информации пересылается один отдельно взятый атрибут насоса.

Постулируется следующее поведение пары клиент - сервер. В обоих случаях (при отправлении и получении информации) сервер ни когда ни чего не ждет, всегда ждет клиент. В случае если сервер занят приемом информации от предыдущего клиента, он перестает принимать следующие запросы, и следующий клиент должен ждать, пока сервер освободится. В случае когда сервер отсылает информацию, клиент должен сам позаботится о том, чтобы успеть обработать или сохранить текущую порцию информации перед приемом следующей.

2. Типы основных потоков данных между GSAU-сервером и GSAU- клиентом

Основными типами потоков данных есть State, StateIndicator (синоним Indicator), Command, CommandIndicator (синоним Indicator). Типы Pump и Position будут использоваться для построения структур обмена и соответствующих конкретных таблиц на основе типа Relation.

Как было постулированно выше, в каждой отдельной операции передачи информации между клиентами и сервером, пересылается один отдельно взятый атрибут насоса. Поэтому тип State есть записью из двух полей:

- pump (натуральное число) - содержит уникальный идентификатор насоса,
- и state (может иметь любой тип из заданного множества типов (TTank, Gas, OnLine и т.д.) - аналог обьединения (Union) языка СИ) каждый из которых описывает соответствующй атрибут насоса. Таким образом, любая переменная типа State однозначно задает значение любого атрибута любого насоса.

Далее, тип Command имеет дополнительный атрибут логического типа, что бы показать должен ли контроллер задать (писать) желаемое значение желаемого атрибута насоса или контроллер должен вычитать (читать) значение это значение желаемого атрибута.

Переменные типа Command и State записываются в соответствующие строки таблиц базы данных ("медленная" связь). Информация о наличии новой команды/новом состоянии передается через "быстрый" канал связи (сеть TCP/IP) переменными типа Indicator, уведомляя об изменениях, произошедших в таблицах БД. Переменная типа Indicator имеет поля, необходимые для надежной работы и однозначного определения, передаваемой информации. Поле index - опередеяет таблицу и колонку в которой произошли изменения, поле id задает строчку этой таблицы, поле get_set - сообщает является ли заданная строчки командой установить или вычитать значения.

Большая часть типов для атрибутов, например тип Gas, является записью (Record) и, одновременно, с определением переменной заданного типа, опередеяет несколько функций, которые могут использоваться вместе с этой переменной - конструктор, деструкторы и реконструкторы. Конструкторы используются для создания записи, деструкторы для доступа к полям, реконструкторы для редактирования полей записи. Имена деструкторов совпадают с именами полей, имена реконструкторов перечисляются после символа <->. Рассмотрим изложенное на примере типа Gas. С определением такого типа (Short Record Definition), задаются следующие функции:

- mk_Gas : Real >< Nat -> Gas - создание записи (конструктор), пара, состоящая из вещественного и натурального числа отображается в запись
- v : Gas -> Real - доступ к полям (деструктор)
- p : Gas -> Nat
- v : Real >< Gas -> Gas - изменение записи (реконструктор)
- p : Nat >< Gas -> Gas

Таким образом, определив переменные или величины как x : Gas, y: Real, z: Nat, мы получаем возможность писать

```
mk_Gas(y,z)
y = v(x) - а не x.v как в большинстве языков программирования
5 < p(x)
x := v(3.141592, x)
p(22222, x)
```

Введем типы данных, специфичные для описания задачи (некоторые универсальные типы описаны в схеме db_start):

```
db_start

scheme Types =
    extend db_start with class
type
    TTank         = {|n : Nat:- n <=2 |}, -- 0 нет бака, 1 и 2 номера баков
    Gas           :: v: Real  <-> v       --  value of gas
                     p: Nat   <-> p,      --  what is position

                   --  типы для описания отдельных характеристик
                   --  используются в структуре, для передачи состояния

                              -- кол-во бензина на заданной кнопке
    OnLine        :: v: Bool  <-> v ,
                              -- есть ли связь с насосом
    Start         :: v: Text  <-> v, -- момент, когда начал считать счетчик
                                     --  total
    Total         :: v: Real  <-> v, -- количестов горючего протекшего
                                     -- через насос
    Measure,                         -- единица измерения
    FreeFill      :: v: Bool  <-> v, -- нужно ли ожидать разрешения наливать
                                     --  горючее


--    Query         :: v: Text  <-> v, -- момент сьема состояния
    Button        :: v: Nat   <-> v, -- агент выбрал кнопку для набора
    NozzleUp      :: v: Bool  <-> v, -- снят ли шланг
    Filling       :: v: Bool  <-> v, -- льется ли горючее
    Filled        :: v: Real  <-> v, -- сколько безина налито в текущем
                                     -- /предыдущем
                                     -- работы цикле
    CycleEnd      :: v: Bool  <-> v, -- разрешение закончить цикл и перейти
                                     --  к  след.
                                     -- горючего
    Paid          :: v: Bool  <-> v, -- горючее уже оплачено кредитной
                                     -- карточкой через устройство на насосе


    FillStart     :: v: Bool  <-> v, -- оператор разрешил наливать горючее
    Limit         :: v: Gas   <->  v, -- оператор задал кол-во горючего и
                                     --  кнопку
    EndWait       :: v: Bool  <-> v, -- ожидает разрешение перейти к
                                     --  следующему циклу работы
    Freeze        :: v: Bool  <-> v, -- работа насоса остановлена
    Restart       :: v: Bool  <-> v, -- перевести насос в начальное
                                     -- состояние


    Error         :: v: Text <-> v,  -- обнаружена программная ошибка

    Proportion    :: p: Nat                    <-> p
                                     -- uid of position/button
                     v: {|n: Nat:- n <= 100|}   <-> v,
```

```
                        -- пропорция смеси горючего на заданной кнопке
    EOG           :: p: Nat        <-> p
                     isGas: Bool   <-> isGas,
                        -- end of gas               на заданной кнопке
    Tank          :: p: Nat         <-> p
                     tank1: TTank   <-> tank1
                     tank2: TTank   <-> tank2,
                        -- два бака горючего        на заданной кнопке
    Mark          :: p: Nat        <-> p
                     name: Text     <-> name,
                        -- марка бензина           на заданной кнопке
    Price         :: p: Nat    <-> p
                     v: Real    <-> v,
                        -- цена бензина            на заданной кнопке


    State1  =           --  состояние насосов. в этой структуре
                        --  записывается один из атрибутов
                        --  насоса

              Gas         | OnLine  |  Total  |  Start    | Freeze   |
              Error       | Restart |  --Query   |
              Proportion  | Filled  | EndWait | CycleEnd  | Filling  |
              EOG         | NozzleUp  | Limit   | FillStart | FreeFill |
              Tank        | Measure | Paid |
              Mark        | Price   | Button
  ,
    State         ::    pump  : Nat     <-> pump  -- uid of pump
                        state : State1  <-> state --
  ,
    Indicator       --  информация, передаваемая от сервера к клиенту
                    --  (и обратно)
                    --  по быстрому каналу связи. получатель использует
                    --  эту информацию для чтения изменения в состоянии
                    --  и отображения этого изменения
                    ::
               messageNo   : Nat   <->
                                messageNo  -- последов.номер сообщения
                                           -- для решения вопроса было
                                           -- ли потеряно некоторое
                                           -- сообщение при передаче
               index       : Nat   <->
                                index      -- число, которое определяет
                                           -- таблицу и колонку из
                                           -- которой читать
               id          : Key  <->
                                id         -- номер строчки, которая
                                           -- изменилась
               get_set     : Bool <->
                                get_set    -- поле имеет различный смысл
                                           -- для индикаторов состояний и
                                           -- индикаторов команд
--
-- I am to lazy to modify rest of  rsl-code
--                 note        : Text <->
--                                  note       -- поле предназначено для

--                                             -- перечи информацией,
--                                             -- неинтерпретируемой
--                                             -- рассматриваемой компонентой
  ,
    StateIndicator = Indicator
                    --   get_set = false => значение поля
                    --                      было потребовано оператором
                    --   get-set = true  => значение поля было вычитано
                    --                      сервером без команды оператора
  ,
    Pump            --  описание насосов в БД и
                    --  параметры насоса  (меняются редко)
                    --
                    --
           ::   uid      : Nat     <->  uid
                onLine   : Bool    <->  onLine
                start    : TDate   <->  start
                total    : Real    <->  total
                measure  : Text    <->  measure
                nPoss    : Nat     <->  nPoss
                                     -- ? сколько кнопок на насосе
                freeFill : Bool    <->  freeFill

                   --  сигналы, поступающие со стороны агента

                button   : Nat     <->  button   -- кнопка на насосе,
                                                 -- выбранная агентом
                nozzleUp : Bool    <->  nozzleUp
                filling  : Bool    <->  filling
                filled   : Real    <->  filled
                paid     : Bool    <->  paid
                endWait  : Bool    <->  endWait

                   -- сигналы, поступающие со стороны оператора

                fillStart: Bool    <->  fillStart
                cycleEnd : Bool    <->  cycleEnd
                setPos   : Nat     <->  setPos
                setLimit : Real    <->  setLimit
                freeze   : Bool    <->  freeze
                restart  : Bool    <->  restart

                   -- это генерируется программой

                error    : Text    <->  error
```

```
                query    : TDate   <->  query
--                name     : Nat     <-> name
   ,
    Position       --  описание кнопок на насосе
           ::   uid      : Nat     <-> uid
                price    : Real    <-> price
                mark     : Text    <-> mark
                tank1    : TTank   <-> tank1
                tank2    : TTank   <-> tank2
                eog      : Bool    <-> eog
                proportion: Nat    <-> proportion

                pump     : Key     <-> pump
--                name     : Text    <-> name
   ,
    Command        --  команды оператора
            =      --  представляют собой  пару брать/установить
               Bool ><       -- false - to get new state of pump
                             -- true  - to set new state of pump
                   -- и что брать/установить. "что" - определяется
               State         --  ранее введенным типом состояние

   ,
    CommandIndicator = Indicator
                   --   get_set = false => оператор потребовал
                   --                      значение поля
                   --   get_set = true  => оператор устанавливает
                   --                      значение поля

                   --
                   -- набор служебных функций приложения
                   --


value
                   --  является ли изменение состояния допустимым?
                   --  функция - "guards" [2] для выражения предусловий
                   --  в данной работе не используется
                   --
    isConversionValid : State >< State ->      Bool
   ,                --  функция восстановления состояние насоса
    StateCorrection   : State >< State ->
                                               Command
   ,
    pStt : StateIndicator,    -- предыдущее состояние
    rcvCnt: Int    -- recovery count - счетчик попыток восстановления

end
```

3. Глобальные обьекты проекта

Авторы RSL рекомендуют после определения схемы (файл с rsl-спецификациями) Types, содержащий типы и атрибуты для не динамических обьектов, на ее основе определить глобальный обьект T:

```
Types
object T : Types
```

После чего ссылаться на введенные типы и атрибуты в форме T.xxx. Например, для описания прототипа функции get из раздела "Определение функций интерфейса с базой данных" используется следующий текст:

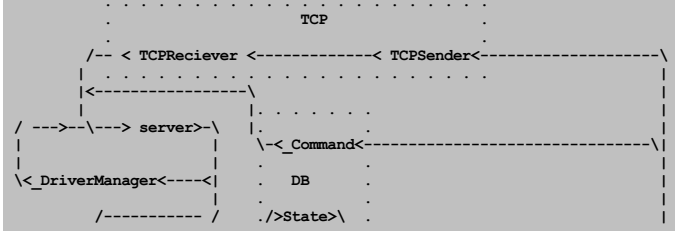
```
get:        T.CommandIndicator      ->
                write
                   Pump.any, Position.any,
                   Cmd4Pmp.any,  Cmd4Pos.any
                T.Command
```

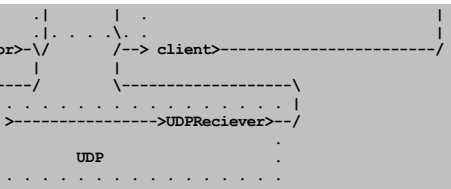
Запись T.Command означает, что всюду определенная (total function, символ ->, в отличие от частично определенных функций, символ -~->) функция get вырабатывает значения типа Command из значений типа CommandIndicator, которые были определены в глобальном обьекте T.

Кроме того, данная функция обладает правом писать в любые атрибуты переменных - обьектов, перечисленных после ключевого слова write: Pump, Position, Cmd4Pmp, Cmd4Pos. Функции с правом доступа к каким - либо переменным также будут называться операторами.

4. Схема взаимодействия между GSAU-сервером и GSAU- клиентом

```
    ver 00.90.10 - some description in DB.rsl have been added

    ver 00.90.00 - improvement of the description

    ver 00.07.06 - the field note have been added to indicator

    ver 00.07.05 - there was a bug in the put - functions

    ver 00.07.04 - a lot if needless let-s have been
                   removed from  DB.rsl
    ver 00.07.03 - phrase check && unique have been added to Relation

    ver 00.07.01
```





Взаимодействие между GSAU - сервером и клиентами заключается в том, что они посылают друг другу индикаторы команд и состояний через каналы TCP/UDP. Получатив индикатор команды или состояния, соответствующая компонента (сервер или клиент) читает необходимую информацию из подходящих таблиц БД.

В первых строчках схемы обьявляется использование схем T и DB. Схема T содержит описание глобального обьекта проекта (обсуждалось выше), схема DB используется для наследования (extend DB) ее свойств текущей схемой GSAU, то есть GSAU порождается из схемы DB.

```
T
,DB
scheme GSAU =
--hide sqlCode   in
extend DB with
class

channel
    TCP : T.CommandIndicator,
    UDP : T.StateIndicator
variable
    msgCnt : Nat := 0
value
                           -- work of server

    Server: Unit    ->
                        write any
                        in TCP    -- канал TCP - вход
                        out UDP   -- канал UDP - выход
                        Unit
    Server() is
        (let v = TCP? in   UDP! server(v)  end
         |=|               -- оператор внешнего выбора
                           -- если в канале TCP есть что читать,
                           -- Server должен прочитать это и отреагировать,
                           -- иначе он может сам вывести информацию
                           -- об изменениях на насосах в канал UDP.
         UDP! server ()  --
                           -- подразумевается, что вызов server()
                           -- отрабатывает быстрее, чем оператор вводит
                           -- новые команды, так же как и вызов client()
                           -- (см. ниже) работает быстрее,
                           -- чем происходят изменения на насосах.
        )
        ; Server()         -- рекурсивный вызов себя - переход
                           -- к следующему циклу работы
   ,
    server : T.CommandIndicator  ->
                        write any
                        T.StateIndicator
                               -- реакция на сигнал от клиента
    server (ci) is             -- сервер отвечает
    PumpMonitor( DriverManager( get(ci)))
   ,

    server : Unit   ->
                         write any
                         T.StateIndicator
    server () is
    PumpMonitor( DriverManager( )) -- сервер генерирует информацию
   ,                               -- без запроса клиента


                                    -- компонента, следящая за состояние
                                    -- системы
    PumpMonitor:  Bool ><           --
                  T.State    -~->
                          write any
                          T.StateIndicator

    PumpMonitor (g_s, s) is
      local
        variable
          newS   : T.State := s,
          cnt    : Int     := 0
      in                            -- попытки восстановления допустимого

                                    -- состояния
        while(T.isConversionValid (newS, get(T.pStt)) = false) do
          let
            (a, b) = DriverManager(T.StateCorrection(newS,
                                                     get(T.pStt)
                                                    )
                                   )
          in
           --  newG_s := a;
             newS   :=  b
          end
          ;
          cnt := cnt + 1;
          if cnt >= T.rcvCnt then
            stop
          end
        end;
        msgCnt := msgCnt + 1;
```

```
      put(g_s, newS, msgCnt)
    end
  ,

                        -- источник возникновения сообщений
                        -- о измененном состоянии насоса / контроллера
                        --

   DriverManager: T.Command  ->    Bool >< T.State,   -- (true, state)
   DriverManager: Unit       ->    Bool >< T.State,   -- (false, state)



                        -- work of client


   Client: Nat    ->                                  -- number of client
                    write any in UDP out TCP
                    Unit
   Client(n) is
       (let v = UDP? in      TCP!  client(v, n)   end
        |=|
        TCP! client(n)
       )
       ; Client(n)
  ,

   client : T.StateIndicator >< Nat ->
                       write any

                       T.CommandIndicator
                                    -- оператор реагирует на изменения
                                    -- в насосах
   client ( si, n) is
   put( UserInterface ( get(si), n), n)
  ,

   client : Nat    ->                       -- оператор выдал команду

                       write any
                       T.CommandIndicator   -- для насосов
   client (n) is
   put( UserInterface ( n ), n)
  ,
                         -- источник возникновения команд
                         -- этой функцией замаскирован оператор
                         -- системы, который выдает команды

   UserInterface: T.State >< Nat    ->    T.Command,
   UserInterface: Nat               ->    T.Command

end
```

Описаны взаимодействие между GSAU - сервером и GSAU - клиентами и алгоритмы работы наиболее крупных компонент. Мы может перейти непосредственно к рассмотрению функций работы с БД (что и есть нашим главным интересом).

5. Описание БД и операторов языка манипулирования данными DML

5.1. Функции высокоуровневого интерфейса

Следующая схема языка RSL в действительности содержит директивы для

1. создания 4 - х таблиц в БД. Эти директивы относятся к группе DDL языка SQL. По - видимому они должны быть реализованы как SQL- скрипт и, вообще говоря, будут выполняться на SQL -сервере в момент инициализации экземпляра БД;
2. создания набора функций для интерфейса между выполняемыми файлами и БД. Мы имеем два выполняемых файла: GSAU- клиент и выполняемый файл, располагающийся на GSAU - сервере - некоторый процесс (все кроме собственно БД). Эти функции должны быть транслированы в язык С.

```
T
,Relation
,TPump
,TPosition
scheme  DB  =
extend  db_start with
class
```

5.2. Определение таблиц базы данных GSAU

В схеме DB приводится описание всех обьектов, представляющих таблицы базы данных, и реляционных связей между ними. Использование параметризованых rsl - схем позволяет описывать ссылочные ограничения целостности. (referencial integrity). Это достигается передачей в качестве актуального параметра второго обьекта - партнера (родителя или ребенка) при описании первого обьекта. Например, в таблице Position есть поле pump, которое ссылается на таблицу Pump, этот факт отражается, тем что в обьект Position (соответствующий таблице Position) передается актуальный параметр (родитель) - обьект Pump (соответствующий таблице Pump), а в обьект Pump передается актуальный параметр (ребенок) - обьект Position.

В обьекте Position параметр Pump используется для проверки наличия первичного ключа для ссылки на таблицу Pump, а в обьекте Pump параметр Position используется для спецификации алгоритма операции delete (on delete restrict). Если в обьекте Position есть строчки, которые ссылаются на первичный ключ p в Pump, то применение операции Pump.delete({p}) приведет к каскадному удалению таких записей в таблице Position.

```
    object

                        -- в эти две таблицы будем
                        -- записывать параметры команд оператора
                        --
       Cmd4Pmp : TPump  (Cmd4Pos)
      , Cmd4Pos : TPosition (Cmd4Pmp)


                        -- State << Pump >< Position
                        -- в эти две таблицы будем записывать
                        -- изменения атрибутов насосов и контроллеров
      , Position: TPosition (Pump)
      , Pump    : TPump  (Position)
```

5.3. Определение функций интерфейса с базой данных GSAU

Все множество функций интерфейса с базой данных представляет собой две пары put и get. Одна пара - положить/взять команду, вторая - положить/взять состояние параметра. Каждая из функций использует соответствующие обьекты базы данных и их функции-члены класса (Insert, Update, Delete, Select).

```
value
                          --  взять  команду

    get:        T.CommandIndicator      ->
                          write
                              Pump.any, Position.any,
                              Cmd4Pmp.any,  Cmd4Pos.any

                          T.Command
    get (ci) is

      let T.mk_Indicator (_, idx, id, g_s) = ci in
        if idx < 100 then     -- будем использовать атрибуты
          let p = if          -- насоса
                    g_s
```

5.3.1. Использование оператора SELECT

Это пример использования оператора SELECT. В случае трансляции этого текста в C++ код, в этом месте можно использовать структура обмена данными (sde). Тогда преобразование кода представляется очевидным. Смотри для примера файл src/gsau_up.cpp, строчка #35

```
GSAU_RCmd4Pmp  Cmd4Pmp  (database);
...
       rc = Cmd4Pos.Select(&id);   // строчка #59
       p  = (GSAU_RPosition *) &Cmd4Pos;
...
```

здесь происходит создание обьекта - структуры обмена данными (sde) Cmd4Pmp и чтение заданной строчки из таблицы SQL серверва в эту структуру обмена. После этого можно использовать поля структуры обмена. Они приняли значения одноименных столбцов таблицы. Пример использования приводится на строчке #196 файла gsau_pu.cpp.

```
                    p->proportion.toStr(),   //
```

Наблюдается полное совпадения. Практически 1 строчка rsl текста преобразуется в 1 строчку C++ кода.

```
                  then
                      Cmd4Pmp.select(id)
                  else
                      Pump.select (id)
                  end
          in              -- после извлечения записи она больше
                          -- не нужна
```

5.3.2. Использование оператора DELETE

Это пример использования оператора DELETE.

```
            if g_s then Cmd4Pmp.delete({id}) end;

                        -- из записи p извлечь атрибут uid и какой

            ( g_s,      -- сделать из них пару
                        -- (признак_команды, состояние)
              T.mk_State (
                  T.uid(p),
                  case idx of
                    1 ->  T.mk_OnLine(T.onLine(p))
                   ,
                    2 ->  T.mk_Start(T.start(p))
                   ,
                    3 ->  T.mk_Total(T.total(p))
                     ,
                    4 ->  T.mk_Measure(T.measure(p))
                   ,
                    5 ->  T.mk_NPos(T.nPos(p))

                   ,
                    6 ->  T.mk_FreeFill(T.freeFill(p))
                   ,
                    7 ->  T.mk_Button(T.button(p))
                   ,
                    8 ->  T.mk_NozzleUp(T.nozzleUp(p))
                   ,
                    9 ->  T.mk_Filling(T.filling(p))
                   ,
                    10 -> T.mk_Filled(T.filled(p))
                   ,
                    11 -> T.mk_CycleEnd(T.cycleEnd(p))
                   ,
                    12 -> T.mk_Paid(T.paid(p))
                   ,
                    13 -> T.mk_FillStart(T.fillStart(p))
                   ,
                    14 -> T.mk_Limit(T.mk_Gas(T.setLimit(p),
                                              T.setPos(p)
                                             )
                                     )
                   ,
                    15 -> T.mk_EndWait(T.endWait(p))
                   ,
                    16 -> T.mk_Freeze(T.freeze(p))
                   ,
                    17 -> T.mk_Restart(T.restart(p))
                   ,
                    18 -> T.mk_Error(T.error(p))
```

```
                end
            )
          )
        end
      else                          -- будем использовать атрибуты
        let p = if                  -- кнопки насоса
                    g_s
                then
                    Cmd4Pos.select(id)
                else
                    Position.select (id)

                end
        in
          if g_s then Cmd4Pos.delete({id}) end;
          ( g_s,
            T.mk_State (
                T.uid(p),     -- !!!!!! эта ошибка  сюда надо ложить
                              --  уид насоса
                case idx of
                              --
                  101 ->  T.mk_Proportion(T.pump(p), T.proportion(p))
                              --      ^^^^
                              --      а сюда уид позиции
                 ,
                  102 ->  T.mk_EOG(T.pump(p), T.eog(p))
                 ,
                  103 ->  T.mk_Tank(T.pump(p),T.tank1(p), T.tank2(p))
                 ,
                  104 ->  T.mk_Mark(T.pump(p), T.mark(p))
                 ,
                  105 ->  T.mk_Price(T.pump(p), T.price(p))
                end
            )
          )
        end
      end
    end
  ,

  put:        T.Command >< Nat ->     -- client number
                        write  Pump.any, Position.any,
                               Cmd4Pmp.any,  Cmd4Pos.any
                        T.CommandIndicator
  put ((g_s, stt), n) is
  local
    variable
      p   : Nat-set := {},
      idx : Nat     := 0
  in
    let T.mk_State(pUid, s) = stt in
  --                          --
      if g_s then             -- в случае команды установить
```

5.3.3. Использование оператора INSERT

Это пример использования оператора INSERT. В этом месте используется перегруженная функция insert, в которую явно передается только актуальный параметр для поля uid. Остальные поля принимают значения по умолчанию.

```
        Cmd4Pmp.insert(     -- создал запись и вставил ее в таблицу
              pUid          -- (ЭТО СПЕЦ. INSERT cm Pump.rsl)

        );
        p := {Cmd4Pmp.identity};-- и взял ее id
        Cmd4Pmp.query(T.timeStamp(),p)
                            -- записал время выдачи команды
      else                  -- для чтения - узнать id существующей
                            -- записи
        p:= Pump.uid2id(pUid)
      end
      ;
      if p~= {} then
        case s of
        T.mk_OnLine(b) ->
```

5.3.4. Использование оператора UPDATE

Это пример использования оператора UPDATE. С этим оператором есть небольшая сложность. В случае rsl - текста нет проблем с производительностью и, как видно ниже, просто выполняется замена значения поля onLine таблицы Cmd4Pmp во всех строчках, первичный ключ которых содержится во множестве p. Если понадобится редактировать второе поле необходимо поступать таким же образом. Это не ДОЛЖНО переносится в C++ текст в такой форме (это будет снижать производительность). Надо поменять значение соответствующих полей структуры обмена данными и потом выполнить операцию update сразу для всех полей записи таблицы (структуры обмена).

```
            if g_s then     -- если задавать значение
                            -- то в новую запись вписать значение
                            --  онлайн
                Cmd4Pmp.onLine (b, p)
            end;             -- иначе ничего не делать
            idx := 1
        ,
         T.mk_Start(t) ->
            if g_s then
                Cmd4Pmp.start (t, p)
            end;
            idx := 2
        ,
         T.mk_Total(t) ->
            if g_s then
                Cmd4Pmp.total (t, p)
            end;
```

```
        idx := 3
,
   T.mk_Measure(t) ->            -- не готово
     if g_s then
         Cmd4Pmp.measure (t, p)
     end;
     idx := 4
,
 T.mk_NPos(t) ->

     if g_s then
         Cmd4Pmp.nPos (t, p)
     end;
     idx := 5
,
   T.mk_FreeFill(t) ->
     if g_s then
         Cmd4Pmp.freeFill (t, p)
     end;
     idx := 6
,                  --
                   -- попытка установить атрибут насоса
                   -- шланг_поднять в тру не несет большого
                   -- смысла, (шланг от этого не подымется)
                   -- наверно в драйвере эта команда будет
                   -- запрещена или будет игнорироваться.
                   -- этоже отностся к NPos и, вообще, к
                   -- атрибутам для сигналов, поступающих
                   -- со стороны атента.
                   -- введено для однообразия и будущего
                   -- развития.
 T.mk_Button(t) ->
     if g_s then
         Cmd4Pmp.button (t, p)
     end;
     idx := 7
,
 T.mk_NozzleUp(t) ->
     if g_s then
         Cmd4Pmp.nozzleUp (t, p)
     end;
     idx := 8
,
 T.mk_Filling(t) ->
     if g_s then
         Cmd4Pmp.filling (t, p)
     end;
     idx := 9
,
 T.mk_Filled(t) ->
     if g_s then
         Cmd4Pmp.filled(t, p)
     end;
     idx := 10
,
 T.mk_CycleEnd(t) ->
     if g_s then
         Cmd4Pmp.cycleEnd (t, p)
     end;
     idx := 11
,
 T.mk_Paid(t) ->
     if g_s then
         Cmd4Pmp.paid (t, p)
     end;
     idx := 12


,
 T.mk_FillStart(t) ->
     if g_s then
         Cmd4Pmp.fillStart (t, p)
     end;
     idx := 13
,
 T.mk_Limit(T.mk_Gas(r,b)) ->
     if g_s then
         Cmd4Pmp.setLimit (r, p);
         Cmd4Pmp.setPos  (b, p)
     end;
     idx := 14
,
 T.mk_EndWait (t) ->
     if g_s then
         Cmd4Pmp.endWait (t, p)
     end;
     idx := 15
,
 T.mk_Freeze(t) ->
     if g_s then
         Cmd4Pmp.freeze (t, p)
     end;
     idx := 16
,
 T.mk_Restart(t) ->
     if g_s then
         Cmd4Pmp.restart (t, p)
     end;
     idx := 17
,
 T.mk_Error(t) ->
     if g_s then
         Cmd4Pmp.error (t, p)
```

```
                end;
                idx := 18
            ,
             T.mk_Proportion (b, v) ->          -- а читаю я правильно
                if g_s then                     -- читаю уид позиции
                    Cmd4Pos.insert(  b, hd p)     -- спец. таблицу
                    ;
                    p := {Cmd4Pos.identity};
                    Cmd4Pos.proportion (v, p)
                else                -- по уиду позиции и id насоса взяли
                                    -- id позиции
                    p := Position.uid_pump2id (b, hd p)
                end;
                idx := 101
            ,
             T.mk_EOG (b, v) ->
                if g_s then          -- ошибочка есть
                    Cmd4Pos.insert( b, hd p)
                    ;
                    p := {Cmd4Pos.identity};
                    Cmd4Pos.eog (v, p)

                else
                    p := Position.uid_pump2id (b, hd p)
                end;
                idx := 102
            ,
             T.mk_Tank (b, t1, t2) ->
                if g_s then
                    Cmd4Pos.insert( b, hd p)
                    ;
                    p := {Cmd4Pos.identity};
                    Cmd4Pos.tank1 (t1, p);
                    Cmd4Pos.tank2 (t2, p)
                else
                    p := Position.uid_pump2id (b, hd p)
                end;
                idx := 103
            ,
             T.mk_Mark (b, v) ->
                if g_s  then
                    Cmd4Pos.insert( b, hd p)
                    ;
                    p := {Cmd4Pos.identity};
                    Cmd4Pos.mark (v, p)
                else
                    p := Position.uid_pump2id (b, hd p)
                end;
                idx := 104
            ,
             T.mk_Price (b, v) ->
                if g_s then
                    Cmd4Pos.insert( b, hd p)
                    ;
                    p := {Cmd4Pos.identity};
                    Cmd4Pos.price (v, p)
                else
                    p := Position.uid_pump2id (b, hd p)
                end;
                idx := 105
             end
          end
          ;
          T.mk_Indicator(0, idx,
                      if p~={} then hd p else T.notFound end, g_s)
        end
      end
     ,
                                -- положить / взять состояние
--
--          the server - client stream
--

      get:        T.StateIndicator      ->
                            write Pump.any,
                                  Position.any
                            T.State
      get(si) is

         let T.mk_Indicator (_, idx, id, _) = si in
           if idx < 100 then
             let p = Pump.select(id) in
               T.mk_State ( T.uid(p),
                   case idx of
                     1  ->   T.mk_OnLine(T.onLine(p))
                    ,
                     2  ->   T.mk_Start (T.start(p))
                    ,
                     3  ->   T.mk_Total (T.total(p))
                    ,
          --         4  ->   T.mk_Measure  (T.measure(p))
          --        ,
          --         5  ->   T.mk_NPos     (T.nPos   (p))
          --        ,
                     6  ->   T.mk_FreeFill (T.freeFill(p))
                    ,
                     7  ->   T.mk_Button(T.button(p))
                    ,
                     8  ->   T.mk_NozzleUp (T.nozzleUp (p))
                    ,
                     9  ->   T.mk_Filling (T.filling (p))
                    ,
```

```
                  10 ->   T.mk_Filled(T.filled(p))
                 ,
                  11 ->   T.mk_CycleEnd (T.cycleEnd (p))
                 ,
                  12 ->   T.mk_Paid (T.paid (p))
                 ,
                  13 ->   T.mk_FillStart (T.fillStart(p))
                 ,
                  14 ->   T.mk_Limit (T.mk_Gas(T.setLimit (p),
                                               T.setPos(p)
                                               )
                                      )
                 ,
                  15 ->   T.mk_EndWait (T.endWait (p))
                 ,
                  16 ->   T.mk_Freeze (T.freeze(p))
                 ,
                  17 ->   T.mk_Restart (T.restart(p))
                 ,
                  18 ->   T.mk_Error (T.error(p))
                end
            )
          end
        else
          let p = Position.select(id) in
            T.mk_State ( T.uid(p),
                case idx of
                  101  ->  T.mk_Proportion (T.pump (p),
                                            T.proportion(p))
                 ,
                  102  ->  T.mk_EOG (T.pump (p), T.eog(p))
                 ,
                  103  ->  T.mk_Tank (T.pump (p), T.tank1(p),

                                                  T.tank2(p))
                 ,
                  104  ->  T.mk_Mark (T.pump (p), T.mark(p))
                 ,
                  105  ->  T.mk_Price (T.pump (p), T.price(p))
                end
            )
          end

        end
      end

  ,
                              --
                              -- вывод состояния в БД
                              -- и создание индикатора состояния
                              --
   put:      Bool >< T.State  >< Nat
             ->               -- выводить состояние можно на всякий
                        write  --State.any,
                              -- надо изменить строчку либо в
                          Pump.any,
                          Position.any
                        T.StateIndicator
                              -- таблице Position, либо в Pump
   put(g_s, stt, cnt) is                          --
   local
     variable
       p   : Nat-set := {},            -- состояние s
       idx : Nat := 0                  -- проапдейтили значение кнопки
   in                                  -- для насоса с id p
     let T.mk_State(pUid, s) = stt in  -- для насоса p получили
                                       -- изменение
       p := Pump.uid2id(pUid) ;
       if p ~= {} then
           Pump.query(T.timeStamp(), p);
           case s of
           T.mk_OnLine(b)    ->  Pump.onLine(b,p) ;
                                 idx := 1
         ,
           T.mk_Start(t)     ->  Pump.start(t,p);
                                 idx := 2
         ,
           T.mk_Total(r)     ->  Pump.total(r,p);
                                 idx := 3
         ,
--           T.mk_Measure() ->  skip  4
--           T.mk_NPos   () ->  skip  5

           T.mk_FreeFill(b) ->  Pump.freeFill (b, p);
                                 idx :=  6
         ,
           T.mk_Button(n) ->
                                 Pump.button(n,p);
                                 idx := 7
         ,

           T.mk_NozzleUp(b) ->  Pump.nozzleUp(b, p);
                                 idx := 8
         ,
           T.mk_Filling(b)  ->  Pump.filling (b, p);
                                 idx :=  9
         ,

           T.mk_Filled(r)   ->  Pump.filled (r, p);
                                 idx := 10
         ,
           T.mk_CycleEnd(b) ->  Pump.cycleEnd (b,p);
```

```
                                idx := 11
            ,
             T.mk_Paid(b)      ->  Pump.paid (b,p);
                                   idx := 12
            ,
             T.mk_FillStart(b)->  Pump.fillStart (b, p);
                                   idx := 13
            ,
             T.mk_Limit(T.mk_Gas(r,b)) ->     -- команда задает два
                                              -- значения
                                              -- кнопку и кол-во горючего
                                   Pump.setLimit (r,p);
                                   Pump.setPos   (b,p);
                                   idx :=  14
            ,
             T.mk_EndWait(b)  ->  Pump.endWait(b,p);
                                   idx := 15
            ,
             T.mk_Freeze(b)   ->  Pump.freeze(b,p);
                                   idx := 16
            ,
             T.mk_Restart(b)   -> Pump.restart(b,p);
                                   idx := 17
            ,
             T.mk_Error(b)    ->   Pump.error(b,p);
                                   idx := 18
            ,
```

5.3.5. Использование оператора UPDATE #2

```
             T.mk_Proportion(b,v) ->
                                p := Position.uid_pump2id(b, hd p);
                                Position.proportion(v, p);
                                idx := 101
                           --
                           -- update Proportion set proportion = :v
                           --     where id in (select id
                           --                  from Proportion
                           --                  where  uid = :b and
                           --                         pump = :p

                           --                )
                           --
            ,
             T.mk_EOG(b, v)   ->
                                p := Position.uid_pump2id (b, hd p);
                                Position.eog(v,  p);
                                idx := 102
            ,
             T.mk_Tank(b, t1, t2) ->
                                p := Position.uid_pump2id(b, hd p);
                                Position.tank1(t1,  p);
                                Position.tank2(t2,  p);
                                idx := 103
            ,
             T.mk_Mark(b,v)   ->
                                p := Position.uid_pump2id(b, hd p);
                                Position.mark(v,   p);
                                idx :=  104
            ,
             T.mk_Price(b, v) ->
                                p :=  Position.uid_pump2id(b, hd p);
                                Position.price( v,  p);
                                idx := 105
--          ,
--           _             ->      skip
             end
          end;
          T.mk_Indicator(cnt,
                           idx,
                            if p~={}then hd p else T.notFound end,
                             g_s)
       end
    end
end
```

5.4. Описание таблицы типа TPump.

Этот текст непосредственно должен конвертироваться в SQL-скрипт генерации БД. Так как RSL по умолчанию не создает функции редактирования полей записи, то для полей, которые можно редактировать в таблице, необходимо задать эти функции. Мы задаем такие функции для каждого поля записи. Они являются аналогом SQL - оператора UPDATE Pump SET xxxx=yyyyy WHERE id in {ks}

> *Примечание*
> *Возможно, что так как в SQL редактирование разрешено по умолчанию, то надо создать новый тип в RSL (или как - то параметризовать Short Record Definition - тип языка RSL, который используется для описания таблиц сейчас), для которого генерировать эти функции автоматически. И, только, в случае явного запрета редактировать поля таблицы не генерировать их.*

Тип создается на основе типа обьявленного в RSL - схеме Relation. Схема параметризуется подчиненной таблицей типа TPosition. Показано, что в SQL-скрипте создания подчиненной таблицы должно появится обьявление внешнего ключа внешнего ключа (ссылочная целостность), ссылающегося на поле id главной таблицы типа TPump. Кроме того, операция удаления строчки в главной таблице должна вызывать каскадное удаление строчек в подчиненной таблице.

```
T
,Relation                       -- ST - slave table
scheme TPump  ( ST: Relation(T{Position for record}) )
    =
                                -- запись record заменил на запись Pump
                                -- теперь можно редактировать поля
    extend   hide hideDelete in   use hideDelete for delete in
                Relation (T {Pump for record} ) with
                                --
                                --
    class
```

```
    value
        --
        --  функции редактирования полей (аналог update)
        --

        uid    : Nat  >< Key-set -> write tbl, sqlCode Unit
        uid    (b, ks) is set_int  (b, ks, T.uid)
       ,
        onLine : Bool >< Key-set -> write tbl, sqlCode Unit
        onLine (b, ks) is set_bool (b, ks, T.onLine)
       ,
        start  : Text >< Key-set -> write tbl, sqlCode Unit
        start  (b, ks) is set_text (b, ks, T.start)
       ,
        total  : Real >< Key-set -> write tbl, sqlCode Unit
        total  (b, ks) is set_real (b, ks, T.total)
       ,
        measure : Text >< Key-set -> write tbl, sqlCode Unit
        measure (b, ks) is set_text (b, ks, T.measure)
       ,
        nPoss   : Nat >< Key-set -> write tbl, sqlCode Unit
        nPoss   (b, ks) is set_int (b, ks, T.nPoss)
       ,
        freeFill: Bool >< Key-set -> write tbl, sqlCode Unit
        freeFill(b, ks) is set_bool (b, ks, T.freeFill)
       ,


        query  : Text >< Key-set -> write tbl, sqlCode Unit
        query  (b, ks) is set_text (b, ks, T.query)
       ,
        button  : Nat >< Key-set -> write tbl, sqlCode Unit
        button  (b, ks) is set_int (b, ks, T.button)
       ,

        nozzleUp  : Bool >< Key-set -> write tbl, sqlCode Unit
        nozzleUp  (b, ks) is set_bool (b, ks, T.nozzleUp)
       ,
        filling  : Bool >< Key-set -> write tbl, sqlCode Unit
        filling (b, ks) is set_bool (b, ks, T.filling)
       ,
        filled  : Real >< Key-set -> write tbl, sqlCode Unit
        filled  (b, ks) is set_real (b, ks, T.filled)
       ,
        cycleEnd : Bool >< Key-set -> write tbl, sqlCode Unit
        cycleEnd(b, ks) is set_bool (b, ks, T.cycleEnd)
       ,
        paid     : Bool >< Key-set -> write tbl, sqlCode Unit
        paid    (b, ks) is set_bool (b, ks, T.paid    )
       ,



        fillStart: Bool >< Key-set -> write tbl, sqlCode Unit
        fillStart(b, ks) is set_bool (b, ks, T.fillStart)
       ,
        setPos  : Nat >< Key-set -> write tbl, sqlCode Unit
        setPos  (b, ks) is set_int (b, ks, T.setPos)
       ,
        setLimit: Real >< Key-set -> write tbl, sqlCode Unit
        setLimit(b, ks) is set_real (b, ks, T.setLimit)
       ,
        endWait : Bool >< Key-set -> write tbl, sqlCode Unit
        endWait (b, ks) is set_bool (b, ks, T.endWait)
       ,
        freeze  : Bool >< Key-set -> write tbl, sqlCode Unit
        freeze  (b, ks) is set_bool (b, ks, T.freeze)
       ,
        restart : Bool >< Key-set -> write tbl, sqlCode Unit
        restart (b, ks) is set_bool (b, ks, T.restart)
       ,

        error   : Text >< Key-set -> write tbl, sqlCode Unit
        error   (b, ks) is set_text (b, ks, T.error)
       ,
        --
        -- поля для оператора Update закончились
        --
```

Необходимо уметь задавать умолчательные значения поля таблицы и уметь показать эти значения программисту.

Перегруженная функция insert содержит эти значения.

Функция uid2id является примером применения функции where обьекта Relation. Актуальным параметр для where есть функция -\p:T.Pump:- T.uid(p) = u, которая имеет прототип T.Pump -> Bool и сравнивает поле uid записи типа T.Pump со значением u.

```
        insert  : Nat -> write tbl, sqlCode, identity Unit

        insert  (uid) is
            insert ( T.mk_Pump (uid, false, timeStamp(), 0.0,
                                                  "litre", 0, false,
                               0, false, false, 0.0, false, false,
                               false, false, 0, 0.0, false, false,
                               "", timeStamp()
                              )
                   )
          insert  : Nat -> write tbl, sqlCode, identity Unit

        insert  (uid) is
            insert ( T.mk_Pump (uid, false, timeStamp(), 0.0,
                                                  "litre", 0, false,
                               0, false, false, 0.0, false, false,
```

```
                            false, false, 0, 0.0, false, false,
                            "", timeStamp()
                           )
                )
      ,
       uid2id : Nat -> write tbl, sqlCode
                       Key-set
       uid2id ( u) is where ( -\p:T.Pump:- T.uid(p) = u )
```

Перегруженная функция delete должна выполнять каскадное удаление () записей в подчиненной таблице. Перегруженная операция delete

```
       alter table Position
         add  constraint pos_pmp_fk
         foreign key (pump) references Pump(id)
         ON DELETE CASCADE;
```

является часть этого оператора alter Table. В действительности она задает фразу "ON DELETE CASCADE".

> *Примечание*
> *Таким способом можно моделировать любые триггеры, которые нужно задать на таблице. Очевидно, что за универсальность языка RSL приходится расплачиваться большим количеством строчек. Текст на специализированном языке SQL, естественно, получается короче.*

```
      ,
       delete :  Key-set               ->  write ST.any, tbl, sqlCode
                                  Unit

       delete (kPmps) is        --  delete from ST where pump in :kPmps

             ST.delete(  {kPos | kPos: T.Key      :-
                                 T.pump(ST.tbl(kPos)) isin kPmps
                         }
                    );


             hideDelete(kPmps)

   axiom
```

Аксиома задает ограничение UNIQUE для поля uid.

```
     [uid_unique]
       unique (-\ (x,y): T.Pump >< T.Pump :- T.uid (x) = T.uid(y))
--       all x, y: Key :-
--                 {x,y} <<= dom tbl  /\ T.uid(tbl(x)) = T.uid(tbl(y))
--                     =>

--                       x = y
    ,


     [measure]
       check (-\ r: T.Pump :- T.measure(r) isin { "litre", "gallon"} )



   end
```

5.5. Описание таблицы типа TPosition.

Параметризация нужна, что бы показать зависимость этой таблицы от другой. Актуальный параметр используется в аксиоме primary_pump.

```
T
,Relation
scheme TPosition ( R:  class type
                              record,
                              relation = Nat -m-> record
                           variable
                              tbl : relation
                       end
                )
     =

   extend  Relation ( T {Position for record} ) with
                              -- запись record заменил на запись Pump
                              -- теперь можно редактировать поля
   class
   value
       uid    : Nat  >< Key-set -> write tbl, sqlCode Unit
       uid    (b, ks) is set_int  (b, ks, T.uid)
      ,
       price  : Real >< Key-set -> write tbl, sqlCode Unit
       price  (b, ks) is set_real (b, ks, T.price)
      ,
       mark  : Text >< Key-set -> write tbl, sqlCode Unit
       mark  (b, ks) is set_text (b, ks, T.mark)
      ,
       tank1   : T.TTank >< Key-set -> write tbl, sqlCode Unit
       tank1   (b, ks) is set_int (b, ks, T.tank1)
      ,
       tank2   : T.TTank >< Key-set -> write tbl, sqlCode Unit
       tank2   (b, ks) is set_int (b, ks, T.tank2)
      ,
       eog : Bool >< Key-set -> write tbl, sqlCode Unit
       eog (b, ks) is set_bool (b, ks, T.eog)

      ,
       proportion   : Nat >< Key-set -> write tbl, sqlCode Unit
       proportion   (b, ks) is set_int (b, ks, T.proportion)
      ,

--        это поле я не разрешаю редактировать после создания
```

```
--        pump   : Nat >< Key-set -> write tbl, sqlCode Unit
--        pump   (b, ks) is set_int (b, ks, T.pump)
--       ,

--        uid2id : Nat -> write tbl, sqlCode        -- лишняя функция
--                         Key-set
--        uid2id ( u) is where ( -\p:T.Position:- T.uid(p) = u )
--       ,
        insert  : Nat >< Key -> write tbl, sqlCode, identity Unit
        insert  (uid, p) is
            insert ( T.mk_Position (
                           uid, 0.0, "", 0, 0, false, 0, p
                       )
                   )
      ,

        uid_pump2id : Nat >< Nat -> write tbl, sqlCode
                        Key-set
        uid_pump2id ( u, pmp) is
            where ( -\p:T.Position:-
                       T.uid(p) = u /\ T.pump(p) = pmp
                  )
       ,
        foo    : Nat -> write  tbl ,  sqlCode   Unit
    axiom

      [uid_pump_unique]   -- (uid, pump ) is unique
        unique (-\ (x,y): T.Position >< T.Position :-
                                T.uid (x) = T.uid(y) /\
                                T.pump(x) = T.pump(y)
                )

--        all x, y: Key :-
--                  {x,y} <<= dom tbl  /\
--                            T.uid(tbl(x)) = T.uid(tbl(y)) /\
--                            T.pump(tbl(x)) = T.pump(tbl(y))
--                        =>
--                        x = y
     ,
```

Аксиома задает большую часть будущего DDL - оператора

```
        alter table Position
          add  constraint pos_pmp_fk
          foreign key (pump) references Pump(id)
          ON DELETE CASCADE;
```

В действительности она задает все кроме фразы "ON DELETE CASCADE".

```
      [primary_pump]      --  pump must be exist
      all pos  : T.Position :- pos isin rng tbl
                       =>
                       T.pump(pos) isin dom R.tbl

    end
```

5.6. Описание нового типа данных Relation.

На произвольном типе данных (type record) задана абстрактная таблица как отображение из целых чисел в record. Схема используется для описания конкретных таблиц через механизм "extend Relation with" - аналог наследования.

Введены операции insert, delete, select, update. Операция update изменяет всю запись целиком.

Функция where представляет собой аналог фразы WHERE оператора SELECT.

Функции check и unique представляют собой аналоги соответствующих фраз CHECK и UNIQUE оператора CREATE TABLE.

Функции set_xxxx будут использоваться в функциях обновления поля позже, когда запись начнет иметь поля (то есть на основе абстрактного типа Relation будет построен конкретный тип для некоторой таблицы, например, TPump).

```
db_start
--,rel

scheme Relation ( R:  class type record  end) =
    hide hDefined in
    extend db_start with
    class
        type
            record   = R.record,
            relation = Nat -m-> record
        variable
            tbl      : relation
        value
            garbage: record,
                                -- функция для проверки ограничений
                                -- целостности

            check : (record -> Bool) -> read tbl Bool
            check (p) is
               ( all x : Key :- x isin dom tbl => p(tbl(x)) )
          ,
                                -- функция для проверки ограничения
                                -- unique

            unique : ((record >< record) -> Bool) -> read tbl Bool
            unique (p) is

               ( all x, y  : Key :- {x,y} <<= dom tbl  =>
                         ( p(tbl(x), tbl(y)) => x = y )
               )
          ,

                                -- аналог операции Insert
```

```
 insert :  Key >< record   ->  write tbl, sqlCode, identity
                               Unit
 insert (k, r) is
     if ~hDefined(k) then
         tbl := tbl !! [k +> r];
         sqlCode  :=Ok;
         identity := k
     else
         sqlCode  := Err;
         identity := k
     end
,
 insert :         record   ->  write tbl, sqlCode, identity
                               Unit
 insert (   r) is  insert (autoInc(dom tbl), r)
,

                         --  аналог операции Delete

 delete :  Key-set           ->  write tbl, sqlCode
                               Unit
 delete (k) is
     if hDefined(k) then
         tbl     := tbl \ k;
         sqlCode :=  Ok
     else
         sqlCode := notFound
     end
,

 delete :  Unit           ->     write tbl, sqlCode
                               Unit
 delete () is
     delete (dom tbl)
,

                         -- аналог  операции Update

 update :  Key >< record   -> write tbl, sqlCode Unit
 update (k, r)  is
     if hDefined(k) then
         tbl := tbl !! [k +> r];
         sqlCode  :=Ok
     else
         sqlCode  := Err
     end
,
                         -- аналог операции Select


 select:  Key               ->   read tbl
                                 write sqlCode
                               record
 select (k)  is
     if hDefined (k) then
         sqlCode := Ok;
         tbl(k)
     else
         sqlCode := notFound;
         garbage
     end
,
                         -- вспомогательная
                         -- функция where используется для
                         -- построения функций, отображающих
                         -- некоторые наборы полей в первичные
                         -- ключи. может использовться как
                         -- фраза where языка SQL

 where: (record -> Bool)  -> read tbl
                               Key-set
 where (p) is
     {x| x: Key :- p(tbl(x))}
,

 hDefined: Key             ->
                               read tbl
                               Bool
 hDefined (k) is k isin dom tbl
,
 hDefined: Key-set         ->
                               read tbl
                               Bool
 hDefined (k) is    k inter dom tbl  ~= {}
,


 set_text : Text >< Key >< (Text >< record -> record)
                           ->   write tbl, sqlCode
                                Unit
                                 -- s_f - функция, которая
                                 --       меняет поле в записи
 set_text  (fld, k,  s_f)  is
             update(k,           --  заменить запись на новую
                 s_f(fld,        --  изменить поле
                     select(k)   --  взять запись
                 )
             )
,

 set_text : Text >< Key-set >< (Text >< record -> record)
```

```
                                        ->    write tbl, sqlCode
                                              Unit
          set_text  (fld, ks, s_f)  is
              if ks ~= {} then
                let k = hd ks in

                  set_text (fld, k, s_f);
                  set_text (fld, ks \ {k}, s_f)
                end
              end
        ,
          set_int : Int >< Key >< (Int >< record -> record)
                                    ->   write tbl, sqlCode
                                         Unit
          set_int  (fld, k, s_f)  is
                     update(k, s_f(fld, select(k)))
        ,
          set_int : Int >< Key-set >< (Int >< record -> record)
                                    ->   write tbl, sqlCode
                                         Unit
          set_int  (fld, ks, s_f) is
              if ks ~= {} then
                let k = hd ks in
                  set_int (fld, k, s_f);
                  set_int (fld, ks \ {k}, s_f)
                end
              end
        ,

          set_real : Real >< Key >< (Real >< record -> record)
                                    ->  write tbl, sqlCode
                                        Unit
          set_real  (fld, k, s_f)  is
                     update(k, s_f(fld, select(k)))
        ,
          set_real : Real >< Key-set >< (Real >< record -> record)
                                    ->   write tbl, sqlCode
                                         Unit
          set_real  (fld, ks, s_f) is
              if ks ~= {} then
                let k = hd ks in
                  set_real (fld, k, s_f);
                  set_real (fld, ks \ {k}, s_f)
                end
              end
        ,
          set_char : Char >< Key >< (Char >< record -> record)
                                    ->  write tbl, sqlCode
                                        Unit
          set_char  (fld, k, s_f)  is
                     update(k, s_f(fld, select(k)))
        ,
          set_char : Char >< Key-set >< (Char >< record -> record)
                                    ->   write tbl, sqlCode
                                         Unit
          set_char (fld, ks, s_f) is
              if ks ~= {} then
                  let k = hd ks in
                      set_char (fld, k, s_f);
                      set_char(fld, ks\{k}, s_f)
                  end
              end


        ,
          set_bool : Bool >< Key >< (Bool >< record -> record)
                                    ->
                                      write tbl, sqlCode
                                      Unit
          set_bool  (fld, k, s_f)  is
                     update(k, s_f(fld, select(k)))
        ,
          set_bool : Bool >< Key-set >< (Bool >< record -> record)
                                    ->   write tbl, sqlCode
                                         Unit
          set_bool  (fld, ks, s_f) is
              if ks ~= {} then
                  let k = hd ks in
                      set_bool (fld, k, s_f);
                      set_bool(fld, ks\{k}, s_f)
                  end
              end
    end
```

НЕКОТОРЫЕ ВЫВОДЫ И РЕЗУЛЬТАТЫ

В завершение, поддерживая тезис о том, что число строк исходного текста, выдаваемое программистом в час, не зависит от языка программирования (см. [9]) , приведем некоторую информацию, о количестве строчек, которое было потрачено, что бы описать GSAU на языке формальных спецификаций RSL и на языках реализации задачи: C++ и SQL. От себя хочется добавить, что количество строчек, которое необходимо написать для решения какой либо задачи, является важной характеристикой языка программирования.

> *Примечание*
> *Если наш подход по поводу использования RSL для описания реляционных баз данных был бы принят, то схемы db_start.rsl и Relation.rsl должны бы быть частью языка RSL. То есть, их не надо было бы учитывать в общей сумме строчек. Кроме того, так как большая часть схемы TPump.rsl, TPosition.rsl является просто заданием функций доступа к полю таблицы, которые должны существовать по умолчанию, то количество строчек в этих схемах было бы в два раза меньше.*

RSL	*клиентская часть (C++)*	*серверная часть (SQL)*	*коментарии*
db_start.rsl:		*38*	*не имеет отношения*
Relation.rsl:		*227*	*к текущей задаче*

```
DB.rsl:            690            16
GSAU.rsl:          184
T.rsl:             38
t_GSAU.rsl:        11
TPosition.rsl:                    113
TPump.rsl:                        210
Types.rsl:         270
                   1193           604       1797

       1797 - 38 - 227 = 1532
            1532 - 150      = 1382              пересчитано в
       604  - 38 - 227 = 339                    соответствие с
            339  - 150      = 189               примечанием


  скрипты генерации БД


Cmd4Pmp.sqs:                      50
Cmd4Pos.sqs:                      46
CRT_ALL.SQS:                      20       +
DROP_STR.SQS:                     25       +
Indicator.sqs:                    29
inidata.sqs:                      45       +
Position.sqs:                     46
Pump.sqs:                         50
types.sqs:                        43



  C++ код, написанный вручную


t_put_get.cpp:     217
dllstart.cpp:      25
gsau_pu.cpp:       438
gsau_up.cpp:       501
gsaugsau.h:        26
include2.c:        4
uid.cpp:           32
gsau.h:            146
gsau_stt.h:        15
gsaudef.h:         96
mkSDE.h:           17
put_get.h:         9
                   1526     +       354   =  1880

  C++ код, выход утилиты генерации структур обмена

Cmd4Pmp:           45
Cmd4Pmp.cpp:       128
Cmd4Pos:           32
Cmd4Pos.cpp:       63
Indicator:         29
Indicator.cpp:     47
Position:          32

Position.cpp:      63
Pump:              45
Pump.cpp:          128
                   612                        2492    всего
                   612 + 1526 =               2138    только клиентская часть
```

> *Примечание*
> *Далее, необходимо отметить, что в решении задачи GSAU существенную роль играет уникальная высокоуровневая библиотека обмена с базой данных. В случае использования других интерфейсных библиотек, таких как ODBC или Embedde SQL, размер кода вырос бы существенно.*

Итак получаем:

	RSL	реализация
всего	1382	2492
серверная часть	189	354
клиентская часть	1193	2138

Даже в условиях использования библиотеки обмена с базой данных соотношение строчек C++ / RSL приблизительно равно 2.

В тоже время соотнешение SQL / RSL практически равно 1, так как скрипты, отмеченные знаком +, в действительности не имеют отношения к задаче, а есть, так сказать, накладным расходом на создание/удаление структуры БД и ввод начальных значений.

Отсюда можно сделать (вполне очевидный) вывод, что SQL, являясь специализированным языком весьма высокого уровня и позволяющий писать эффективные приложения, является уникальным явлением в мире программирования.

В качестве второго примера приведем статистику полученную при реализации проекта Harbour (cm. [5])

Этот пример является образцом гораздо более абстрактной спецификации, которая не учитывает особенности проектирования БД для РСУБД и не отвечает на такие вопросы, интересующие SQL - программиста, как:

- какие алгоритмы относятся к серверной части (SQL- скриптам),
- какие к клиентской части (C++) и т.д.

Спецификация (RSL - код) - 147 строчек, реализация клиента (C- код) - 799 строчек + (хедеры) 207 строчек, реализация сервера (SQL скрипты) - 482 строчки;

Итого 1488 строчек кода. Получаем соотношение 147:1488 - приблизительно 1 к 10.

ЛИТЕРАТУРА

1. C.J. Date. An intpoduction to database systems. Sixth edition. Addison-Wesley Publishing Company.
2. The RAISE Language Group. The RAISE SPECIFICATION LANGUAGE. ISBN 0-13-752833-7. Denmark, Herdfordshire, Prentice Hall Europe, 1992.- 396;
3. Hung Dang Van, Chris George, Tomasz Janowski, and Richard Moore. Specification Case Studies in RAISE. ISBN 1-85233-359-6 Springer, 2002, http://www.iist.unu.edu/RAISE_Case_Studies/;
4. O.G.Piskunov, O. Golovko. Application - SQL server data echange tools. Canadian Intellectual Property Office. Order # 102928, transaction # 250023, 2001;
5. O.G.Piskunov. Harbour - Example of usage RAISE specification method to develop SQL - based application, 10.12.2001, www.i.com.ua/~agp1/software/hrb1.0.tar.gz;
6. Г. Майерс. Надежность программного обеспечения. Перевод с английского Ю.Ю. Галимова, Под редакцией В.Ш. Кауфмана, Издательство "Мир", Москва, 1980;
7. C.B. Jones. Systematic Software Development - Using VDM, 2nd Edition. Prentice-Hall International, 1989;
8. O.G.Piskunov, V.Ilyuhin. Коммуникационная Компонента Системы Управления Заправочной Станцией, 22.11.2002, www.i.com.ua/~agp1/software/gsau0.90.tar.gz;
9. B.W. Boehm, Software Engineering Economics, Prentice Hall, Englewood Cliffs, N.J., 1981;
10. Лутц Прехельт, Эмпирическое сравнение семи языков программирования, Журнал "Открытые системы", #12, 2000, http://www.osp.ru/os/2000/12/045.htm;
11. L. Prechelt and B. Unger, A Controlled Experiment on the Effects of PSP Trimming: Detailed Description and Evaluation, Tech Report 1/1999, Fakultat fur Informattik, Universitat Karlsruhe, Germany, Mar. 1999.
12. V.S. Alagar, K.Periyasamy, Specification of Software System, ISBN 0-387-98430-5, Springer - Verlag, New York, 1998.
13. http://www.cs.concordia.ca/~faculty/alagar