



# Разработка систем реального времени с использованием UML и каркасов приложений

**Дмитрий Рыжов**

**Менеджер по продукту**

**[d.ryzhov@swd.ru](mailto:d.ryzhov@swd.ru)**



- **Распространение языка моделирования UML**
- **Развитие инструментов разработки на основе визуального моделирования**
- **Применение инструментов на всех стадиях процесса разработки**
- **Специализация инструментов**
- **Инструменты для разработки встраиваемых систем и приложений реального времени**
- **Автоматическая генерация кода, тестирование, временной анализ и верификация**

# Пример разработки секундомера

- Секундомер имеет одну кнопку для запуска и остановки и дисплей для отображения. Дисплей отображает минуты и секунды
- При нажатии и отпускании кнопки в течении 2 секунд секундомер запускается либо останавливается
- Если кнопка удерживается нажатой более чем 2 секунды, то секундомер сбрасывается в 0 и останавливается

Rhapsody in C++ by Telelogic - Stopwatch

File Edit View Code Layout Tools Window Help

StopWatch Release

Object Model Diagram: Определение классов и ассоциаций in Stopwatch \*

Двойной рамкой отображаются активные классы, то есть классы работающие в отдельном потоке

Значком в правом верхнем углу отображаются реактивные классы, то есть классы для которых определена диаграмма состояний

Ассоциация между классами приводит к появлению указателя в сгенерированном коде

В сгенерированном коде активные и реактивные классы наследуются от классов каркаса, реализующие данную функциональность

```

classDiagram
    class Button
    class Timer
    class Display
    Button "1" --> "1" Timer : itsTimer
    Timer "1" --> "1" Display : itsDisplay
  
```

```

Button.h
//// Relations and components ////
protected :

    Timer* itsTimer;      /// link itsTimer

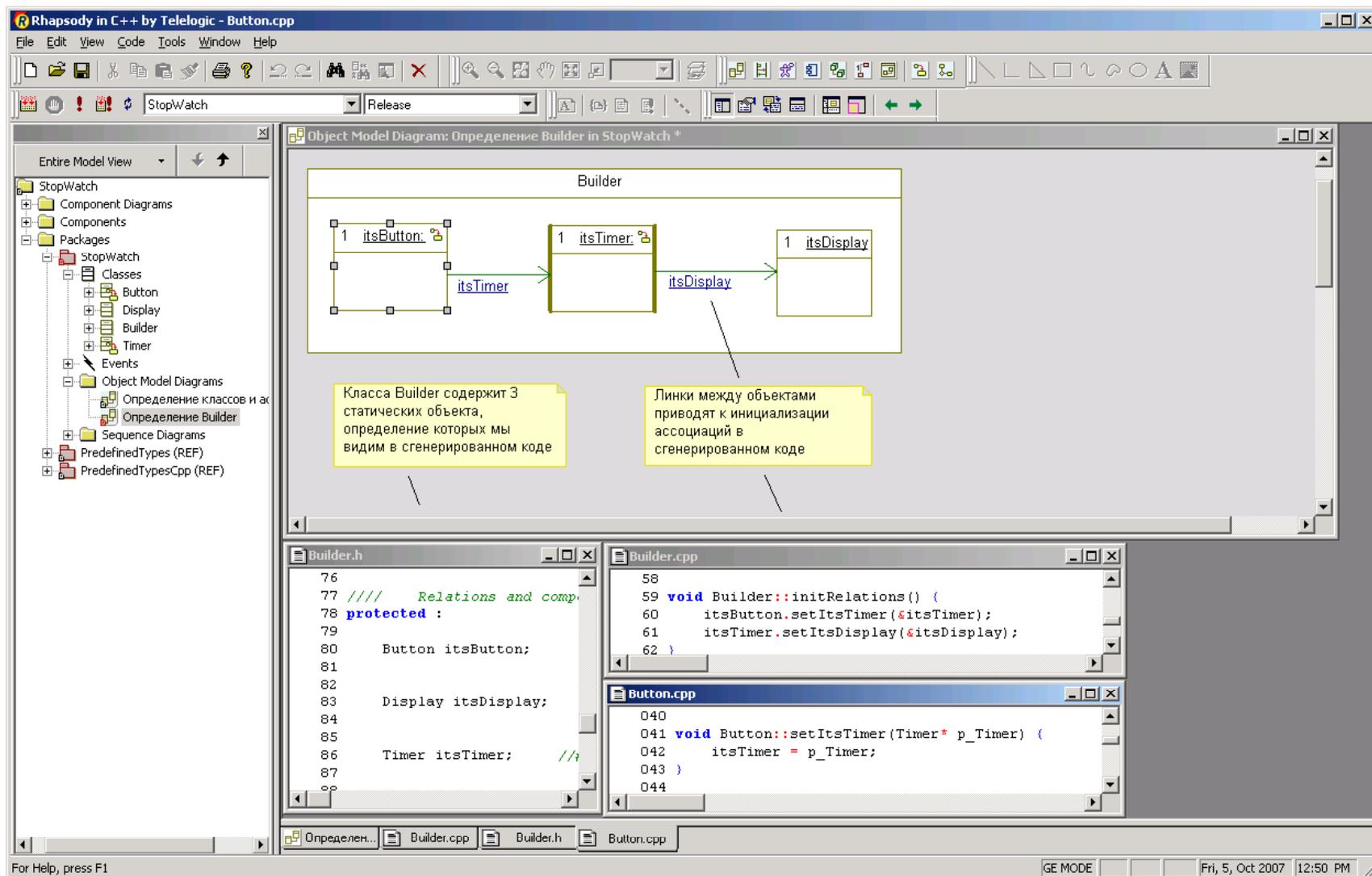
Timer.h
/// class Timer
class Timer : public OMThread, public OMReactive (

//// Constructors and destructors ////
public :
  
```

For Help, press F1

GE MODE

Fri, 5, Oct 2007 1:51 PM



The screenshot shows the Rhapsody in C++ IDE with the following components:

- Object Model Diagram: Определение Builder in Stopwatch \***
  - Class **Builder** contains three static objects: `1 itsButton`, `1 itsTimer`, and `1 itsDisplay`.
  - Association `itsTimer` connects `itsButton` to `itsTimer`.
  - Association `itsDisplay` connects `itsTimer` to `itsDisplay`.
- Annotations:**
  - Класс Builder содержит 3 статических объекта, определение которых мы видим в сгенерированном коде
  - Линки между объектами приводят к инициализации ассоциаций в сгенерированном коде
- Code Editor:**
  - Builder.h** (lines 76-87):

```

76
77 //// Relations and comp.
78 protected :
79
80     Button itsButton;
81
82
83     Display itsDisplay;
84
85
86     Timer itsTimer; ///
87
88

```
  - Builder.cpp** (lines 58-62):

```

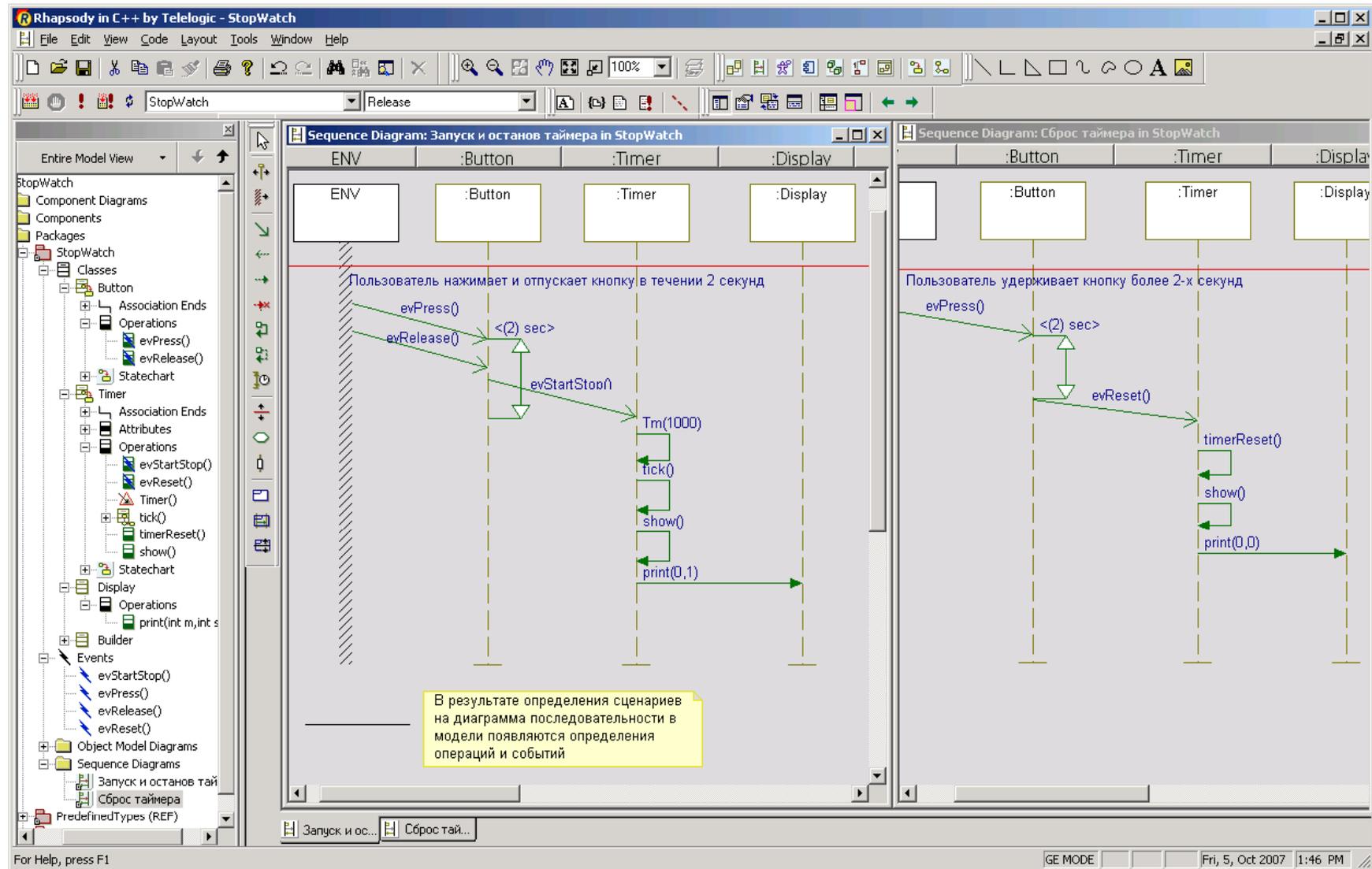
58
59 void Builder::initRelations() {
60     itsButton.setItsTimer(&itsTimer);
61     itsTimer.setItsDisplay(&itsDisplay);
62 }

```
  - Button.cpp** (lines 040-044):

```

040
041 void Button::setItsTimer(Timer* p_Timer) {
042     itsTimer = p_Timer;
043 }
044

```



Rhapsody in C++ by Telelogic - Stopwatch

File Edit View Code Layout Tools Window Help

StopWatch Release

Statechart of: Button

```

    graph LR
      released -- "tm(2000)/  
itsTimer->GEN(evReset);" --> released
      released -- "evPress" --> pressed
      pressed -- "evRelease/  
itsTimer->GEN(evStartStop);" --> released
  
```

Обработка событий главных состояний реализуется в сгенерированном коде методом rootState\_ProcessEvent

Statechart of: Timer

```

    graph LR
      idle -- "evReset" --> Active
      subgraph Active
        idle -- "timerReset();  
show();" --> running
        running -- "evStartStop" --> idle
        running -- "tm(1000)/  
tick();  
show();" --> running
      end
  
```

Обработка вложенных состояний реализуется методом activeState\_processEvent(), который вызывается из rootState processEvent()

Button.cpp

```

IOxfReactive::TakeEventStatus Button::rootState_processEvent() {
    IOxfReactive::TakeEventStatus res = eventNotConsumed;
    switch (rootState_active) {
        case released:
            if (IS_EVENT_TYPE_OF (evPress_StopWatch_id))
            {
                rootState_subState = pressed;
                rootState_active = pressed;
                pressed_timeout = scheduleTimeout(2000, NULL);
                res = eventConsumed;
            }
            break;
        case pressed:
            if (IS_EVENT_TYPE_OF (OMTimeoutEventId))
            {
                if (getCurrentEvent () == pressed_timeout)
                {
                    if (pressed_timeout != NULL)
                    {
                        pressed_timeout->cancel ();
                        pressed_timeout = NULL;
                    }
                    //#[ transition 3
                    itsTimer->GEN (evReset);
                    //#[
                    rootState_subState = released;
                    rootState_active = released;
                    res = eventConsumed;
                }
            }
    }
}
  
```

Timer Button Button.cpp

For Help, press F1

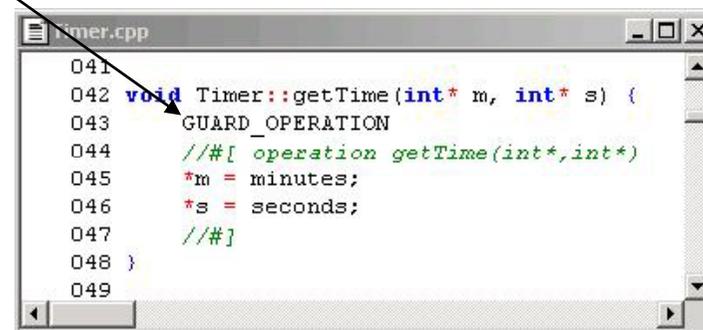
GE MODE Fri, 5, Oct 2007 1:10 PM

- Установка для таймера свойства **Concurrency=guarded**
- В коде появляется макрос **GUARD\_OPERATION**
- Обработка асинхронных событий становится защищенной (**eventGuard != 0**)



```

TakeEventStatus OMReactive::handleEvent(IOxfEvent* ev) {
    TakeEventStatus eventConsumeResult = eventNotConsumed;
    if (eventGuard != 0)
        eventGuard->lock(); // guard against synchronous ev
    eventConsumeResult = processEvent(ev);
    if (eventGuard != 0)
        eventGuard->unlock();
    return eventConsumeResult;
}
    
```



```

041
042 void Timer::getTime(int* m, int* s) {
043     GUARD_OPERATION
044     /*#[ operation getTime(int*,int*)
045     *m = minutes;
046     *s = seconds;
047     /*#]
048 }
049
    
```

The screenshot displays the Rhapsody in C++ IDE interface for a project named 'StopWatch'. The main window is divided into several panes:

- Entire Model View:** A tree view on the left showing the project structure, including 'Component Diagrams', 'Components', 'Configurations', and 'Packages'.
- Component Diagram:** A central diagram showing a 'StopWatch' component. It includes two yellow callout boxes:
  - Top: 'В main файле производится инициализация указанных для компонента классов и запуск OXF каркаса' (Initialization of specified classes for the component and running the OXF framework in the main file).
  - Bottom: 'Для каждого компонента генерируется main и makefile для выбранного SDK' (For each component, a main and makefile are generated for the selected SDK).
- MainStopWatch.cpp:** A code editor on the right showing the source code for the component's main function:
 

```

      ///# configuration Stopwatch::Release
      int main(int argc, char* argv[]) {
          if(OXF::initialize(argc, argv, 6423))
          {
              Builder * p_Builder;
              p_Builder = new Builder;
              p_Builder->startBehavior();
              ///#[ configuration Stopwatch::Release
              ///#]
              OXF::start();
              delete p_Builder;
              return 0;
          }
          else
          {
              return 1;
          }
      }
      
```
- StopWatch (Makefile):** A text editor at the bottom showing the generated Makefile content, including target types and compilation options:
 

```

      ##### Target type (Debug/Release) #####
      #####
      CPPCompileDebug= /Zi /Od /D "_DEBUG" /MDd /Fd "$(TARGET_NAME)"
      CPPCompileRelease= /Ox /D "NDEBUG" /MD /Fd "$(TARGET_NAME)"
      LinkDebug=
      LinkRelease=
      BuildSet=Debug
      SUBSYSTEM=/SUBSYSTEM:console
      COM=False
      RPFrameWorkDll=False
      SimulinkLibName=

      ConfigurationCPPCompileSwitches= /I . /I . /I $(OMROOT)\LangCpp /I $(OMROOT)\
      
```
- Configuration: Release in Stopwatch:** A dialog box on the bottom right showing the configuration for the 'Release' build. It includes tabs for 'Checks', 'Relations', 'Tags', and 'Initialization'. Under 'Initialization', there is a tree view for 'Initial instances' with 'Builder' checked and 'Button', 'Display', and 'Timer' unchecked.

The screenshot displays the Rhapsody in C++ by Telelogic development environment. The main workspace is divided into several panes:

- Statechart of : Button - Builder[0] ->:itsButton:** Shows two states, `released` and `pressed`. Transitions include `evPress` from `released` to `pressed`, and `evRelease/ itsTimer->GEN(evStartStop);` from `pressed` back to `released`. A self-loop on `released` is labeled `tm(2000)/ itsTimer->GEN(evReset);`.
- Statechart of : Timer - Builder[0] ->:itsTimer:** Shows an `Active` state containing `idle` and `running` states. Transitions include `evStartStop` from `idle` to `running` and `evStartStop` from `running` back to `idle`. A self-loop on `running` is labeled `tm(1000)/ tick(); show();`. An `evReset` transition returns to the `Active` state.
- Sequence Diagram: Animated Запуск и останов таймера:** Shows interactions between `ENV`, `:Button`, and `:Timer`. Messages include `Create()`, `evPress()`, `evRelease()`, and `evStartStop()`. A timer `tm(1000) at ROOT.Active running` is shown with `tick()` and `show()` messages.
- Mozilla Firefox:** Displays a web browser window showing a graphical representation of the `Button[0]` object with `evPress` and `evRelease` buttons, each labeled `Activate`. The status bar shows `Готово`.
- Terminal:** Shows the output of the application, displaying a sequence of timestamps from `0:0` to `0:11`, indicating the execution flow.

The status bar at the bottom indicates the system is in `GE MODE` and the date is `Fri, 5, Oct 2007 2:12 PM`.

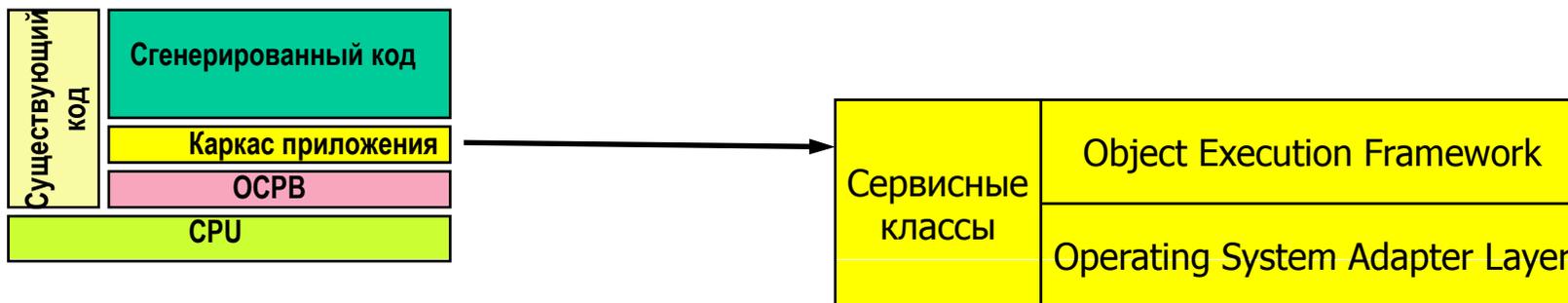
# **Каркас приложения в Telelogic Rhapsody**

- Набор предопределённых взаимодействующих классов
- Предоставляют сервисы при разработке приложений определённого типа
- Разработка приложений путём наследования и переопределения

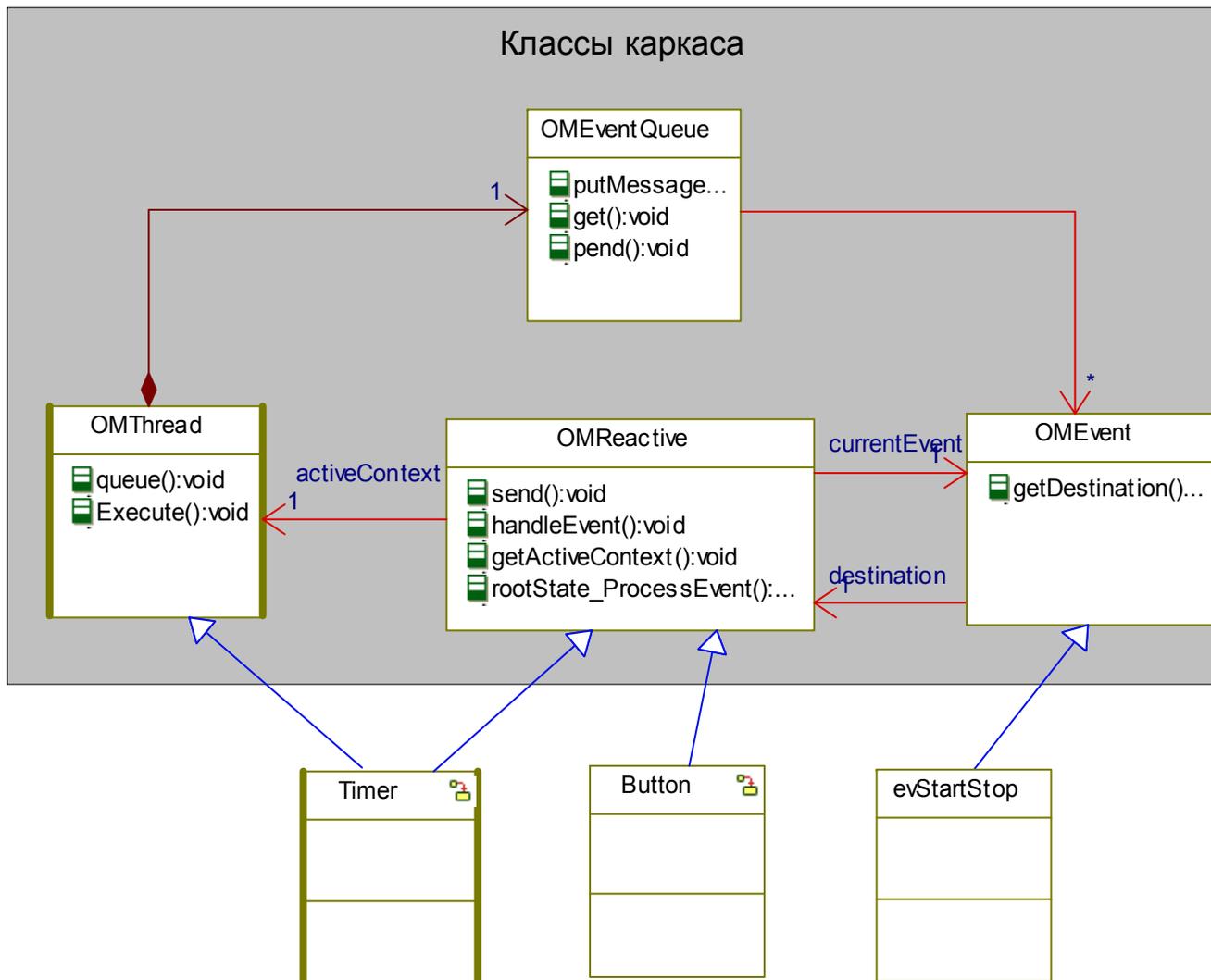
- Отсутствие необходимости создания приложений с нуля
- Определяют архитектуру целевых систем
- Представляют открытые конструкции, могут переопределяться в приложениях

- В сгенерированном коде используется API каркаса
- Каркас реализует основные абстракции приложений реального времени
- Значительная часть функциональности содержится в классах каркаса
- Классы каркаса могут быть адаптированы под конкретные нужды
- Каркас – это библиотека, независимая от генератора кода
- Каркас не ограничивает приложения от использования других библиотек и сервисов ОС

- ➔ Object Execution Framework (OXF)
- ➔ OS adapter level
- ➔ Сервисные классы
- ➔ Animation framework

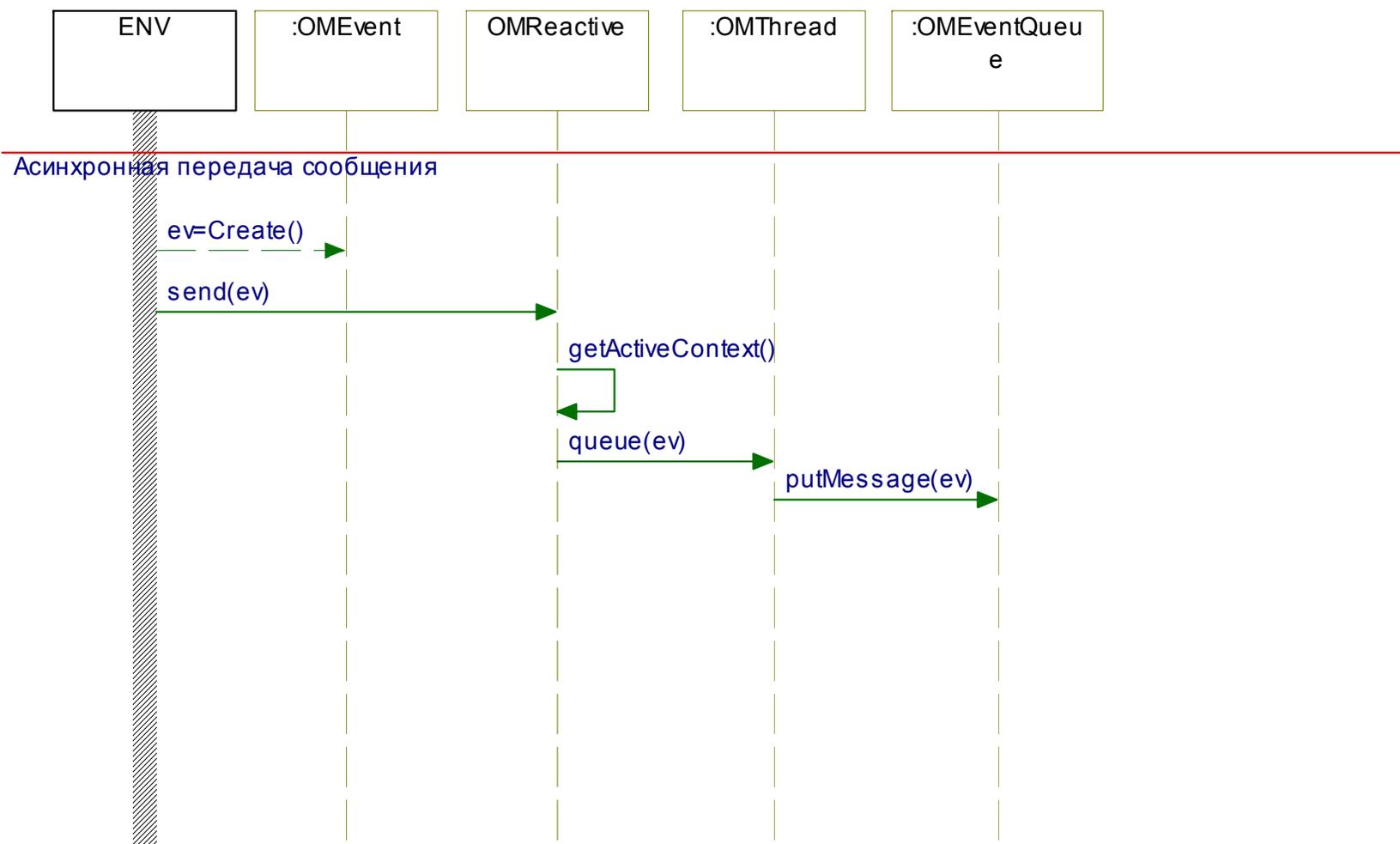


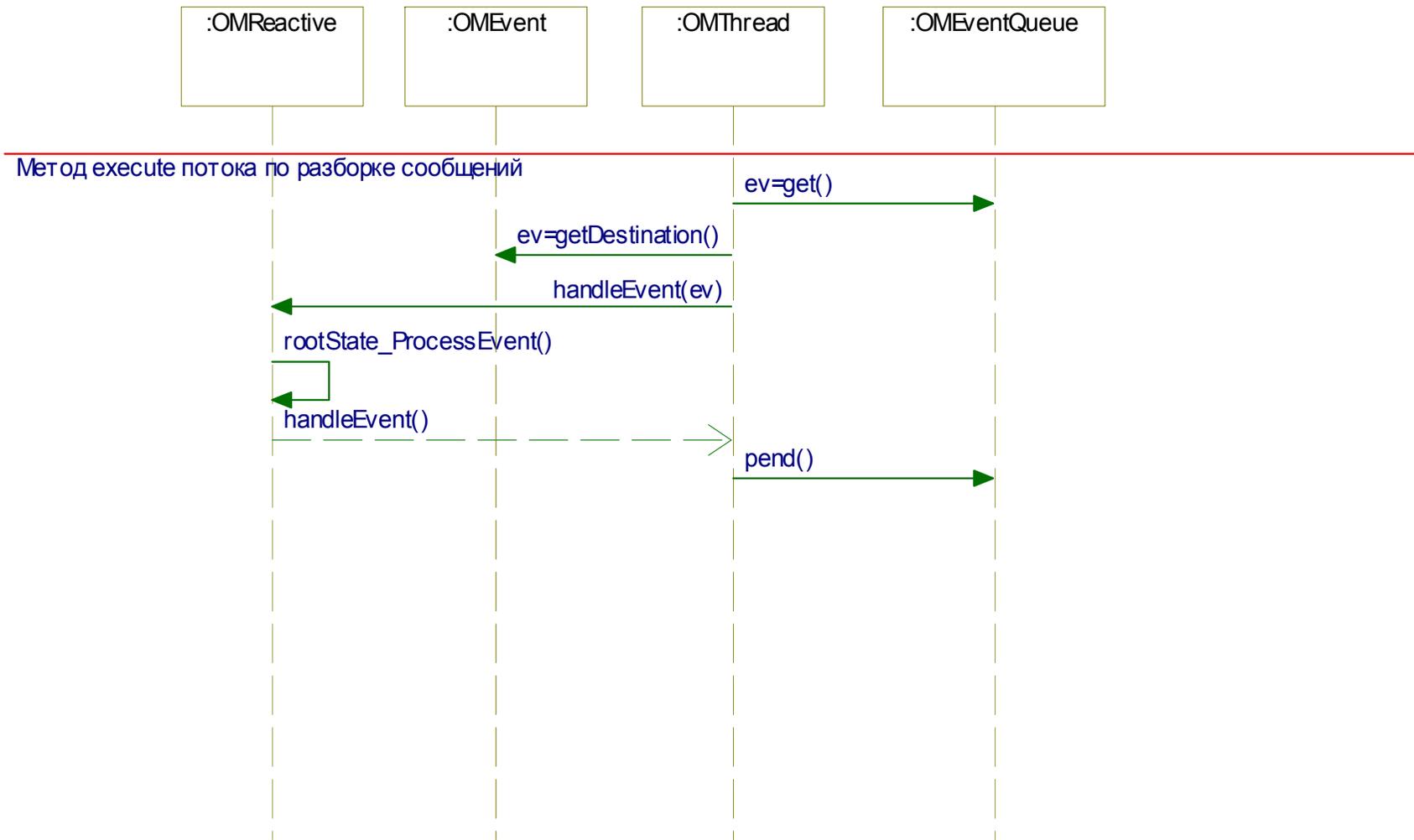
- Асинхронные события
- События времени
- События вызова (синхронные)

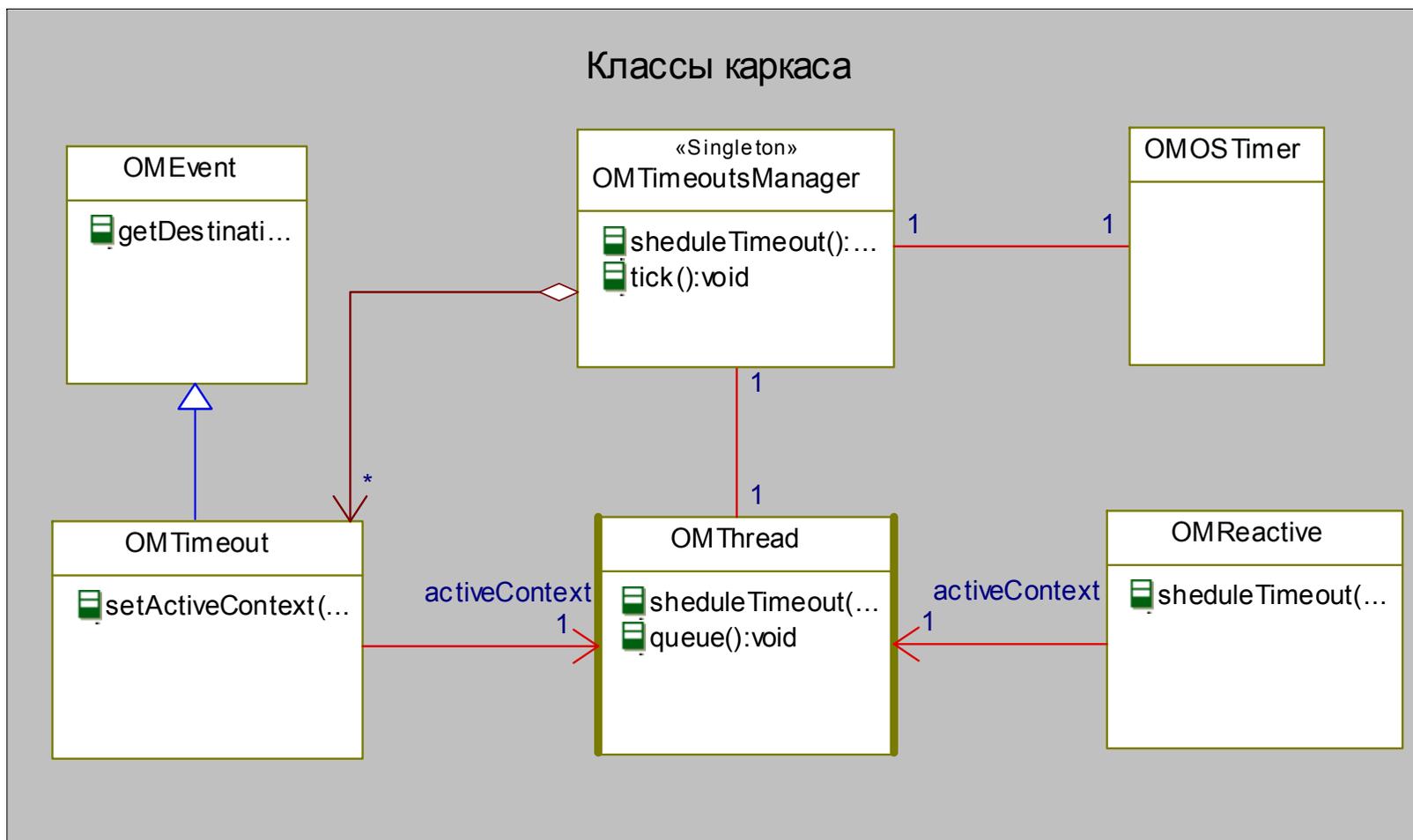


- Наследуется от класса OMThread каркаса
- Запускает в отдельном потоке функцию Execute
- Содержит очередь событий
- Предоставляет функцию queue для помещения событий в очередь
- В Execute разгребает очередь, передавая события адресатам на обработку в функцию handleEvent
- Позволяет перекрыть Execute для реализации другого поведения

- Наследуется от класса OMReactive каркаса
- Предоставляет функцию send для передачи классу асинхронных событий
- Помещает полученные асинхронные события в связанный с ним активный класс для диспетчеризации
- Получает события от активного класса на обработку, вызывающего его функцию handleEvent
- Вызывает виртуальную функцию rootState\_processEvent для обработки событий
- По умолчанию код для функции rootState\_processEvent генерируется на основании диаграммы состояний



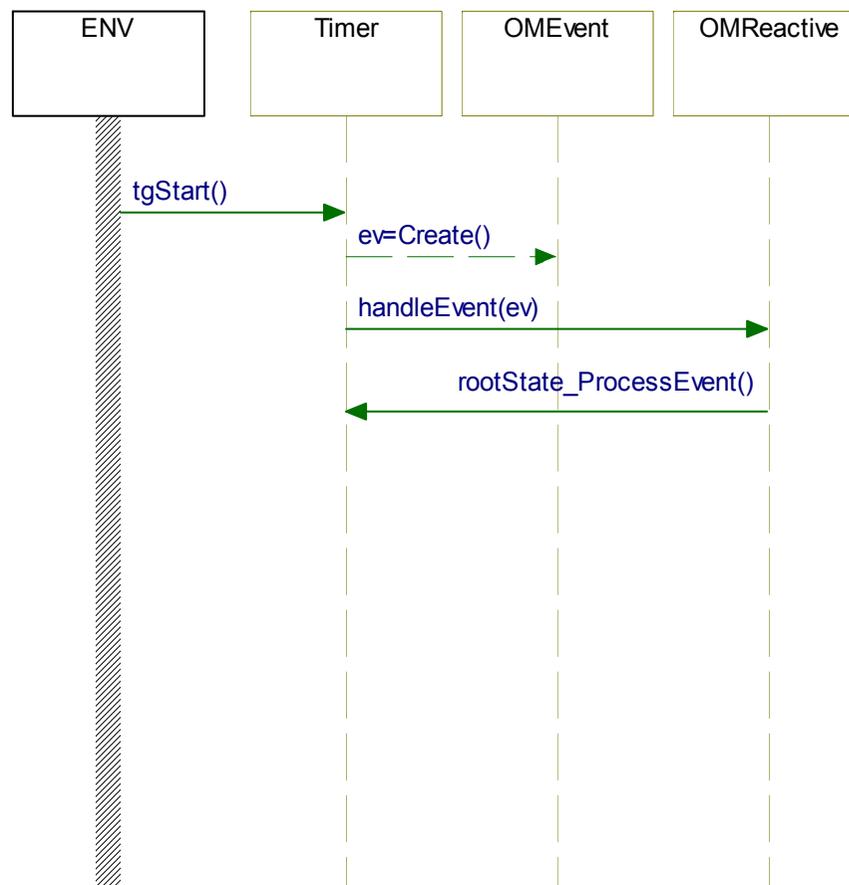


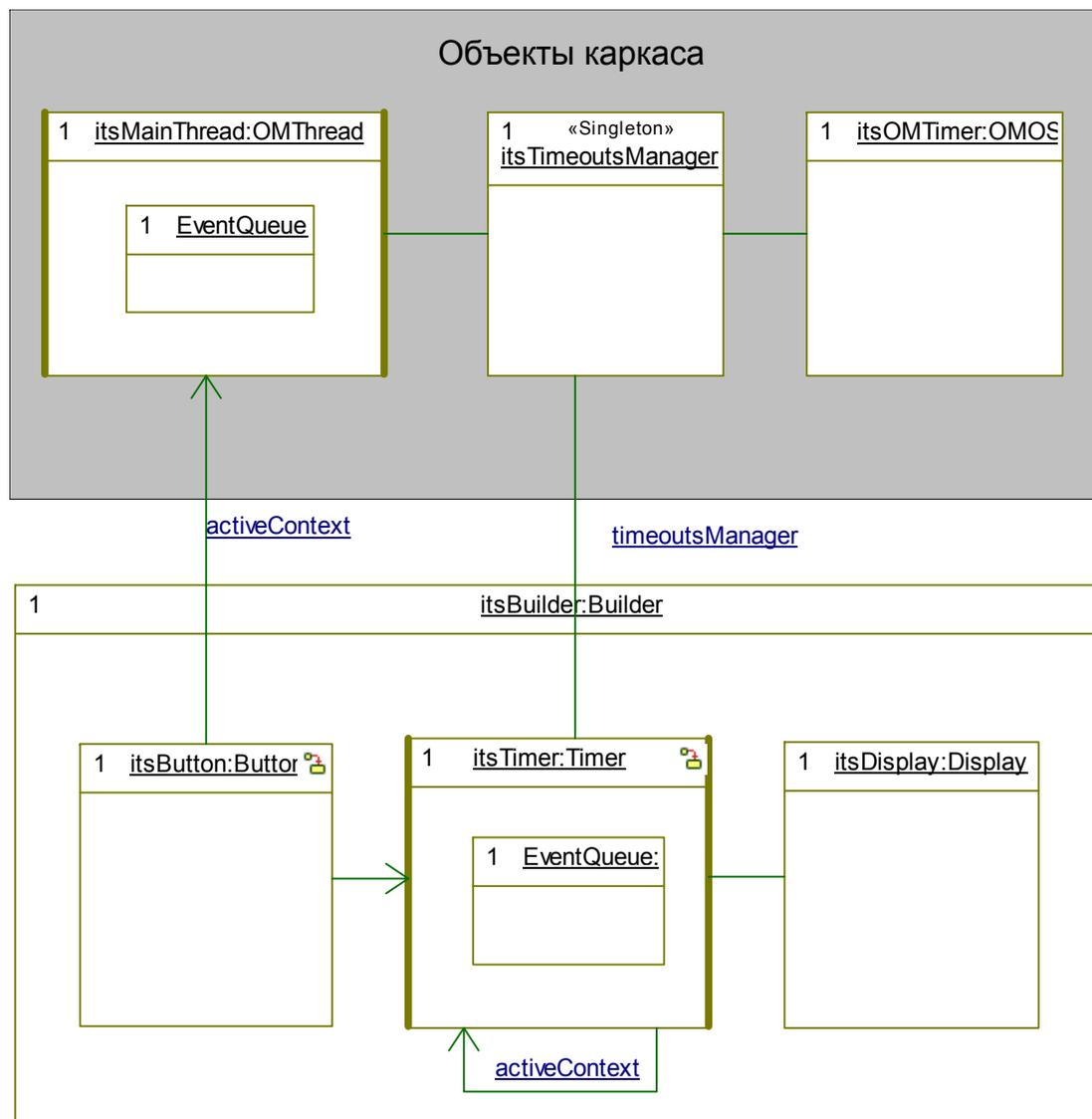


- Таймауты – это особый вид событий на которые можно определять реакции на диаграмме состояний
- Таймауты создаются в сгенерированном коде при входе в состояние и уничтожаются при выходе
- Всеми таймаутами управляет объект `TimeoutsManager`
- При истечении таймаута `TimeoutsManager` помещает его в очередь активного объекта для диспетчеризации
- Таймауты диспетчеризуются активными объектами наравне с другими событиями в очереди



- События вызова генерируются при вызове триггерных операций класса
- В операции создаётся одноимённое событие и сразу же передаётся на обработку в `handleEvent`
- На диаграмме состояний можно определять переходы и реакции на такие события
- Реализация для триггерных операций генерируется автоматически





- По умолчанию связывается с активным объектом `MainThread`
- Связывается с самим собой, если класс объявлен активным
- Связывается с содержащим его активным объектом
- Может быть связан с любым активным объектом путём вызова функции `setActiveContext`

- **Object Execution Framework (использует ОС)**
- **Interrupt Driven Framework (не использует ОС)**
- **Synchronous Framework (не использует ОС)**



Спасибо за внимание!



<http://www.swd.ru/>

196135, г. Санкт-Петербург,  
пр. Юрия Гагарина 23  
тел.: (812) 702-0833

115553, г. Москва,  
пр. Андропова 22/30  
тел.: (495) 780-8831