

Разработка систем реального времени с использованием UML и каркасов приложений

Дмитрий Рыжов
Менеджер по продукту
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 :
  
```

Определени... Button.h Timer.h

For Help, press F1

GE MODE

Fri, 5, Oct 2007 1:51 PM

Rhapsody in C++ by Telelogic - Button.cpp

File Edit View Code Tools Window Help

StopWatch Release

Object Model Diagram: Определение Builder in StopWatch *

Entire Model View

- StopWatch
 - Component Diagrams
 - Components
 - Packages
 - StopWatch
 - Classes
 - Button
 - Display
 - Builder
 - Timer
 - Events
 - Object Model Diagrams
 - Определение классов и ак
 - Определение Builder
 - Sequence Diagrams
 - PredefinedTypes (REF)
 - PredefinedTypesCpp (REF)

Builder

```

classDiagram
    class Builder {
        +Button itsButton
        +Timer itsTimer
        +Display itsDisplay
    }
    Builder --> Button : itsButton
    Builder --> Timer : itsTimer
    Builder --> Display : itsDisplay
  
```

Класса Builder содержит 3 статических объекта, определение которых мы видим в сгенерированном коде

Линки между объектами приводят к инициализации ассоциаций в сгенерированном коде

Builder.h

```

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

```

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

Button.cpp

```

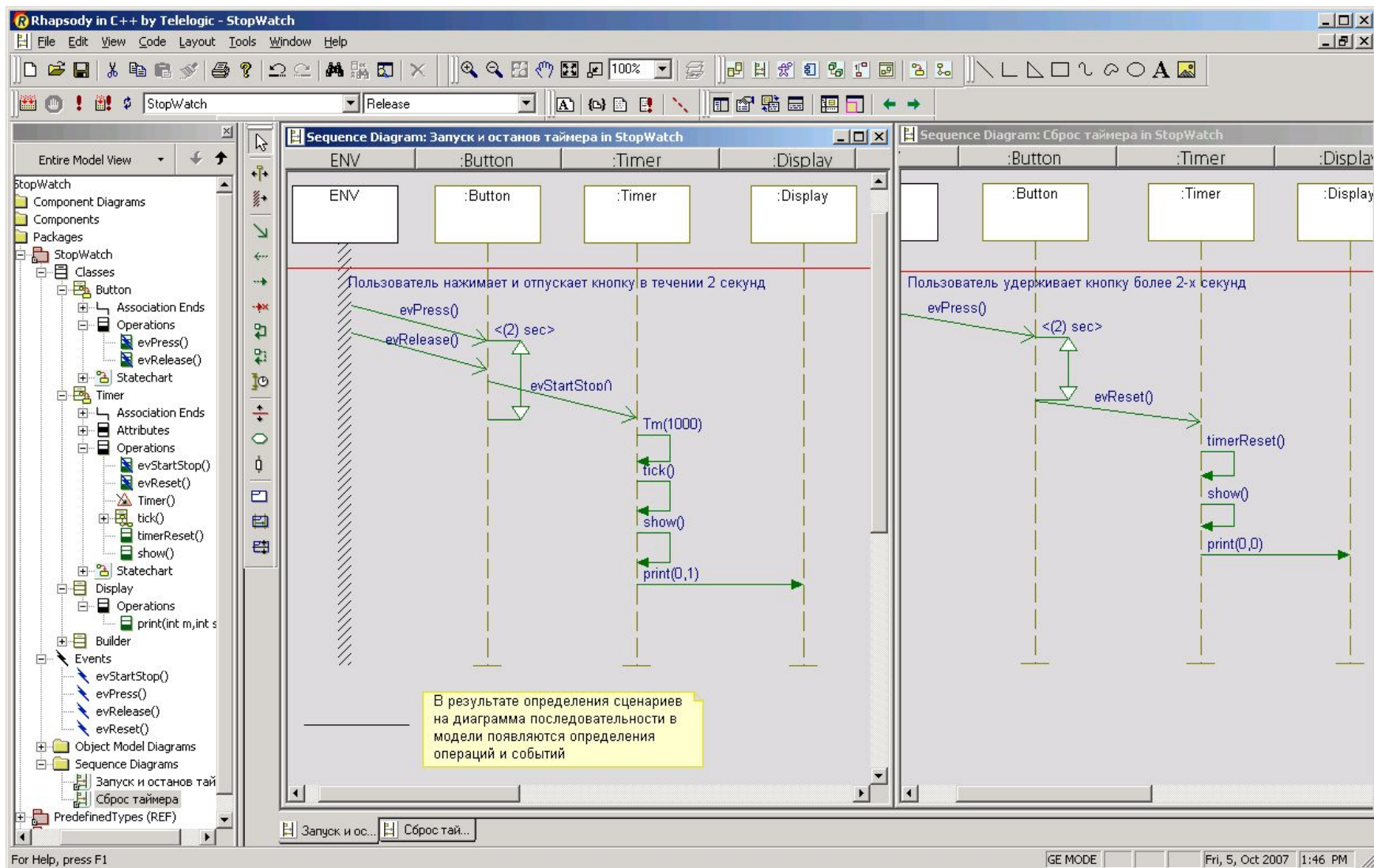
040
041 void Button::setItsTimer(Timer* p_Timer) {
042     itsTimer = p_Timer;
043 }
044
  
```

Определен... Builder.cpp Builder.h Button.cpp

For Help, press F1

GE MODE Fri, 5, Oct 2007 12:50 PM

Определение сценариев для запуска и сброса секундомера



Rhapsody in C++ by Telelogic - Stopwatch

File Edit View Code Layout Tools Window Help

StopWatch Release

Statechart of : Button

tm(2000)/
itsTimer->GEN(evReset);

released pressed

evPress

evRelease/
itsTimer->GEN(evStartStop);

Обработка событий главных состояний реализуется в сгенерированном коде методом rootState_ProcessEvent

Statechart of : Timer

evReset

Active

idle running

/timerReset();
show();

evStartStop

evStartStop

tm(1000)/
tick();
show();

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

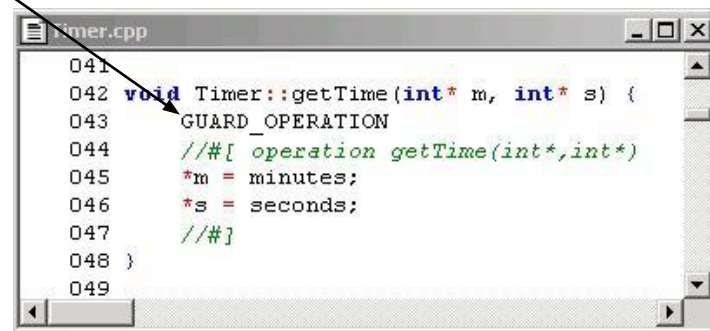
For Help, press F1

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

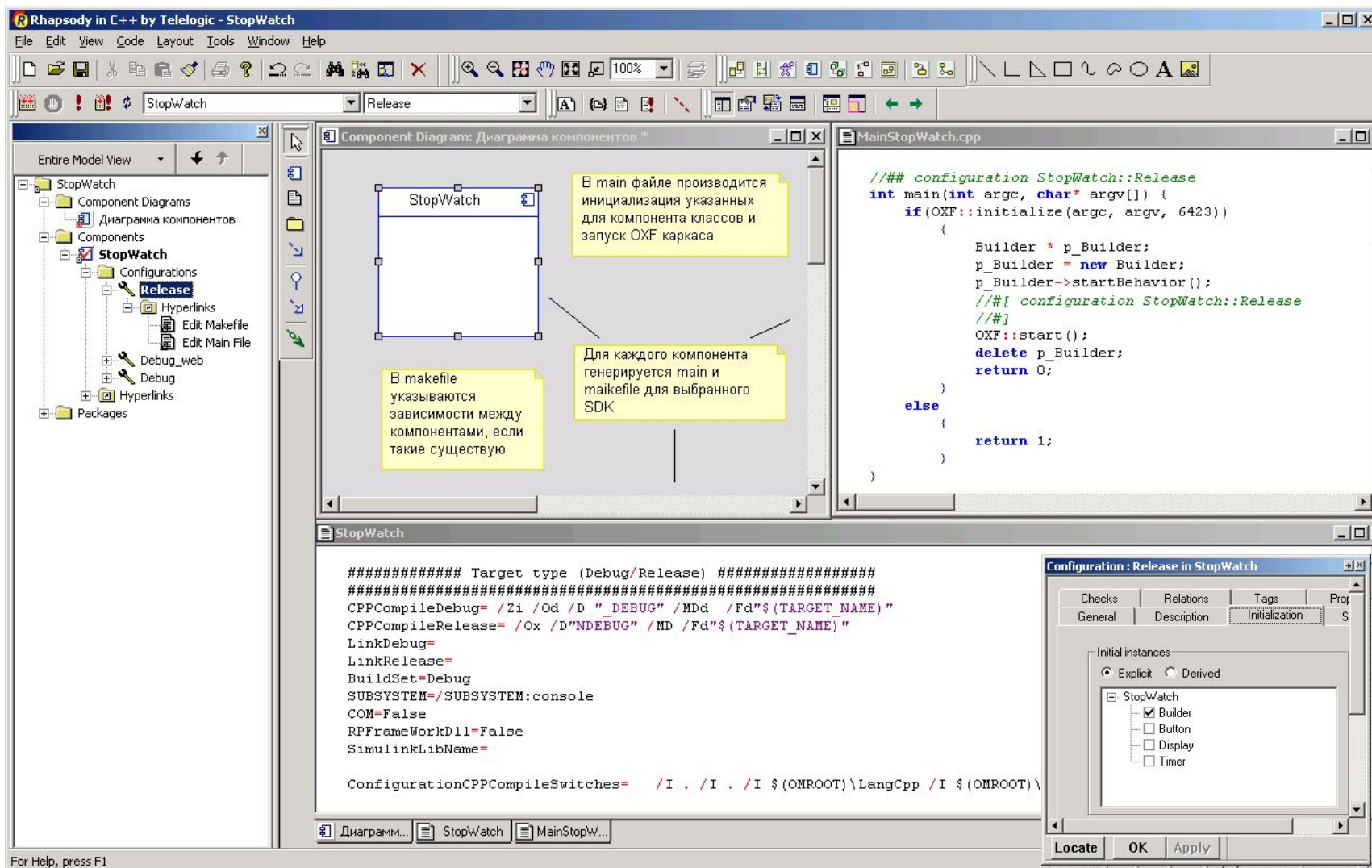
- Установка для таймера свойства **Concurrency=guarded**
- В коде появляется макрос **GUARD_OPERATION**
- Обработка асинхронных событий становится защищенной (`eventGuard != 0`)



```
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
```



Rhapsody in C++ by Telelogic - Stopwatch

File Edit View Code Layout Tools Window Help

StopWatch Release

Component Diagram: Диаграмма компонентов *

StopWatch

В main файле производится инициализация указанных для компонента классов и запуск OXF каркаса

В makefile указываются зависимости между компонентами, если такие существуют

Для каждого компонента генерируется main и makefile для выбранного SDK

MainStopWatch.cpp

```

    ///# 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

```

    ##### 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

Checks Relations Tags Prof

General Description Initialization S

Initial instances

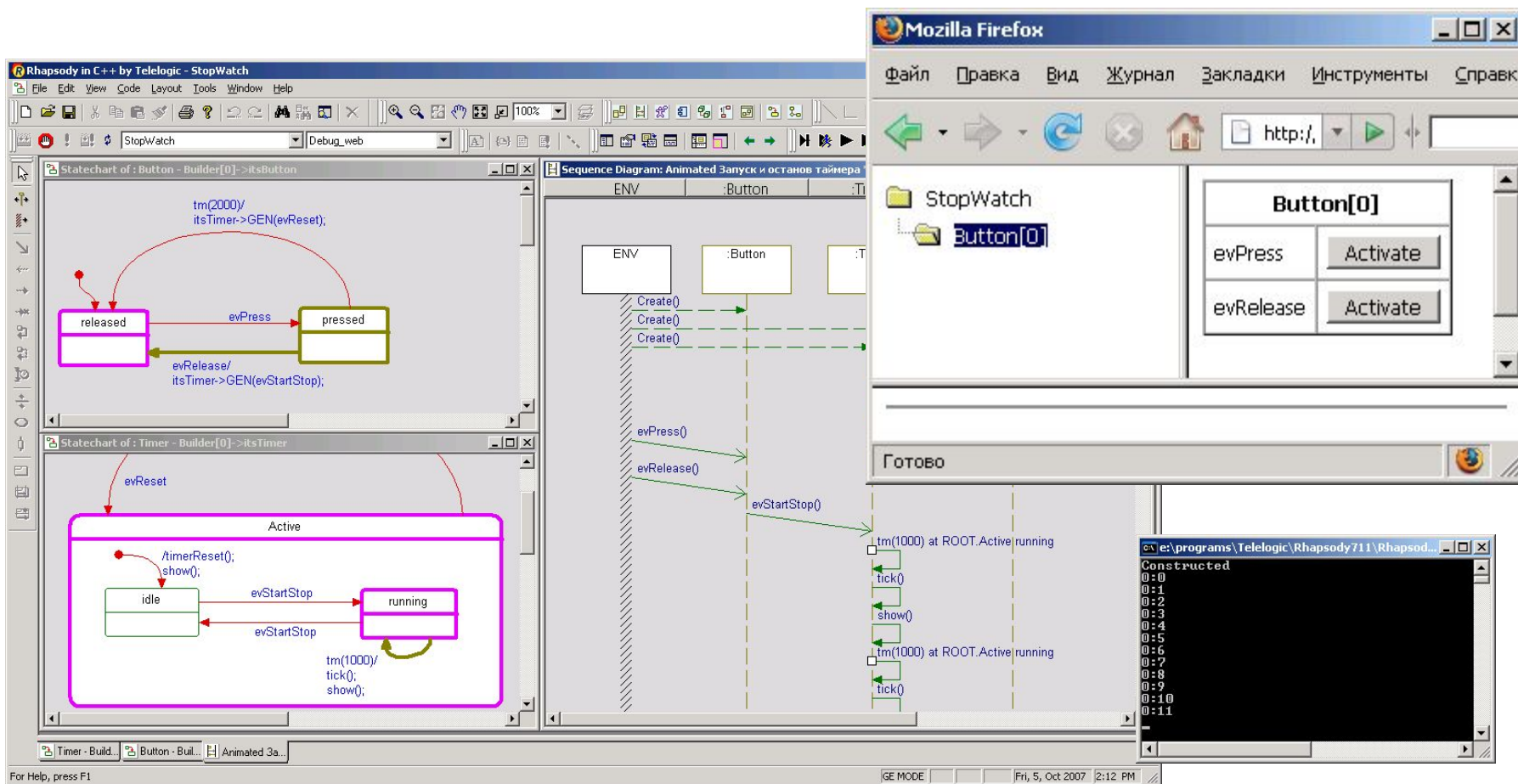
☒ Explicit ☐ Derived

StopWatch

- ☒ Builder
- ☐ Button
- ☐ Display
- ☐ Timer

Locate OK Apply

For Help, press F1



The image displays a development environment for modeling and testing applications at the model level. The main window is **Rhapsody in C++ by Telelogic - Stopwatch**, which contains several views:

- Statechart of :Button - Builder[0]->itsButton:** This statechart 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 the **released** state is labeled **tm(2000)/itsTimer->GEN(evReset)**.
- Statechart of :Timer - Builder[0]->itsTimer:** This statechart shows an **Active** state containing two sub-states, **idle** and **running**. Transitions include **evReset** from idle to running, **evStartStop** from running to idle, and **evStartStop** from idle to running. A self-loop on the **running** state is labeled **tm(1000)/tick(); show();**.
- Sequence Diagram: Animated Запуск и останов таймера:** This diagram illustrates the interaction between the environment (**ENV**), the button (**:Button**), and the timer (**:Timer**). It shows messages like **Create()**, **evPress()**, **evRelease()**, and **evStartStop()** being sent between the objects. The timer's internal state transitions (**tm(1000) at ROOT.Active running**) are also visible.

In the background, a **Mozilla Firefox** browser window is open, displaying the **StopWatch** application. The application interface includes a folder structure with **StopWatch** and **Button[0]**, and a control panel for **Button[0]** with buttons for **evPress** and **evRelease**, each with an **Activate** button. The status bar indicates **Готово** (Ready).

A terminal window in the bottom right corner shows the execution log:

```

e:\programs\Telelogic\Rhapsody711\Rhapsod...
Constructed
0:0
0:1
0:2
0:3
0:4
0:5
0:6
0:7
0:8
0:9
0:10
0:11
  
```

The bottom status bar of the Rhapsody window shows **GE MODE** and the date/time **Fri, 5, Oct 2007 2:12 PM**.

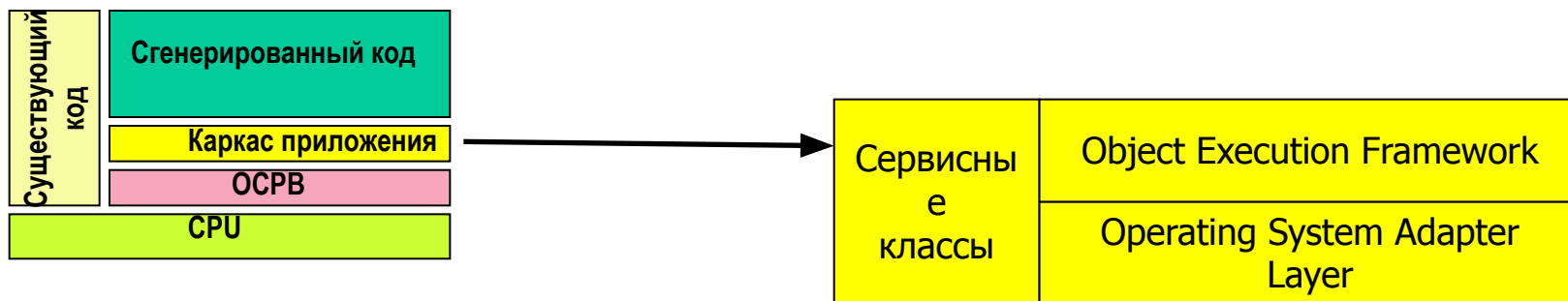
Каркас приложения в Telelogic Rhapsody

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

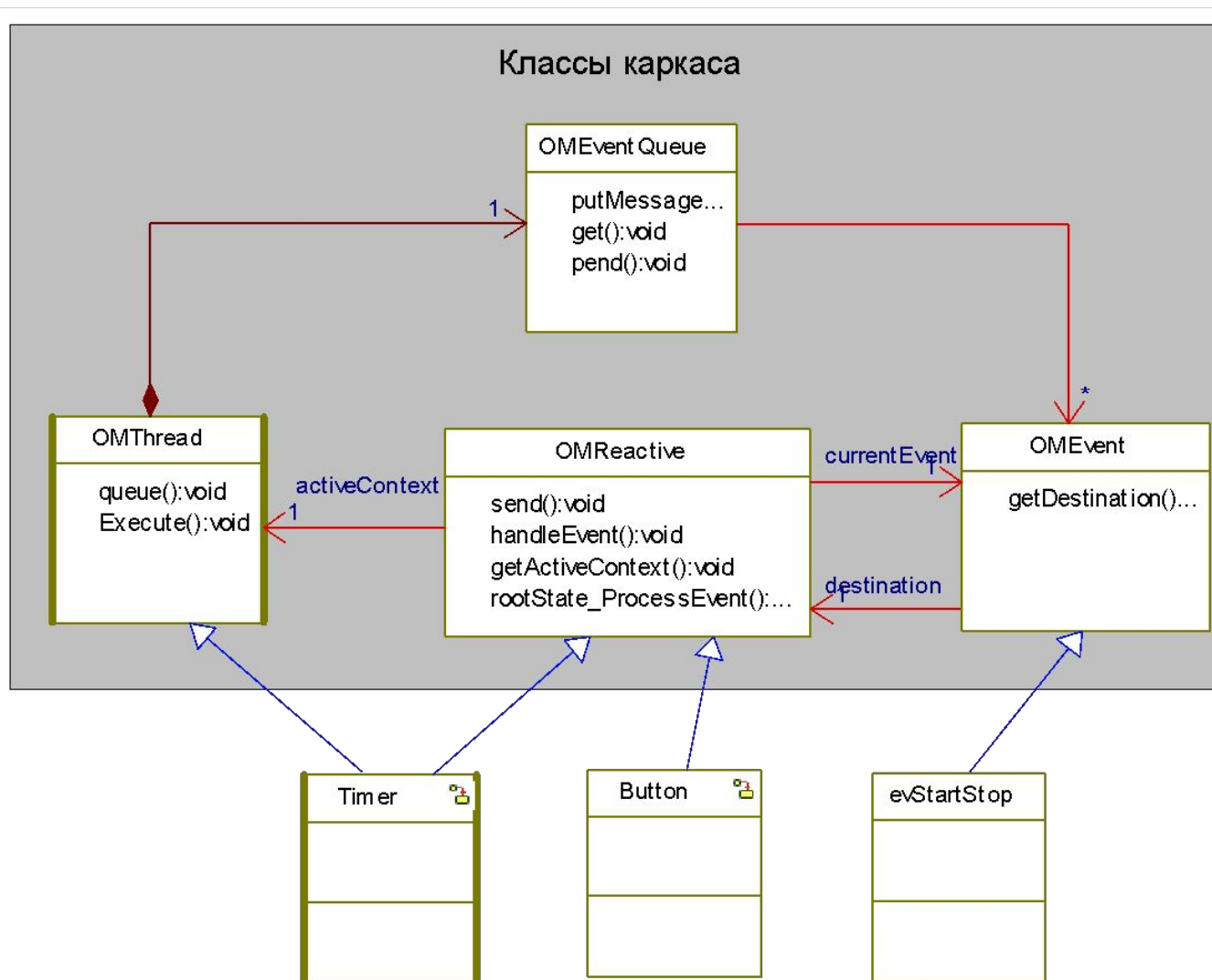
- Отсутствие необходимости создания приложений с нуля
- Определяют архитектуру целевых систем
- Представляют открытые конструкции, могут переопределяться в приложениях

- 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

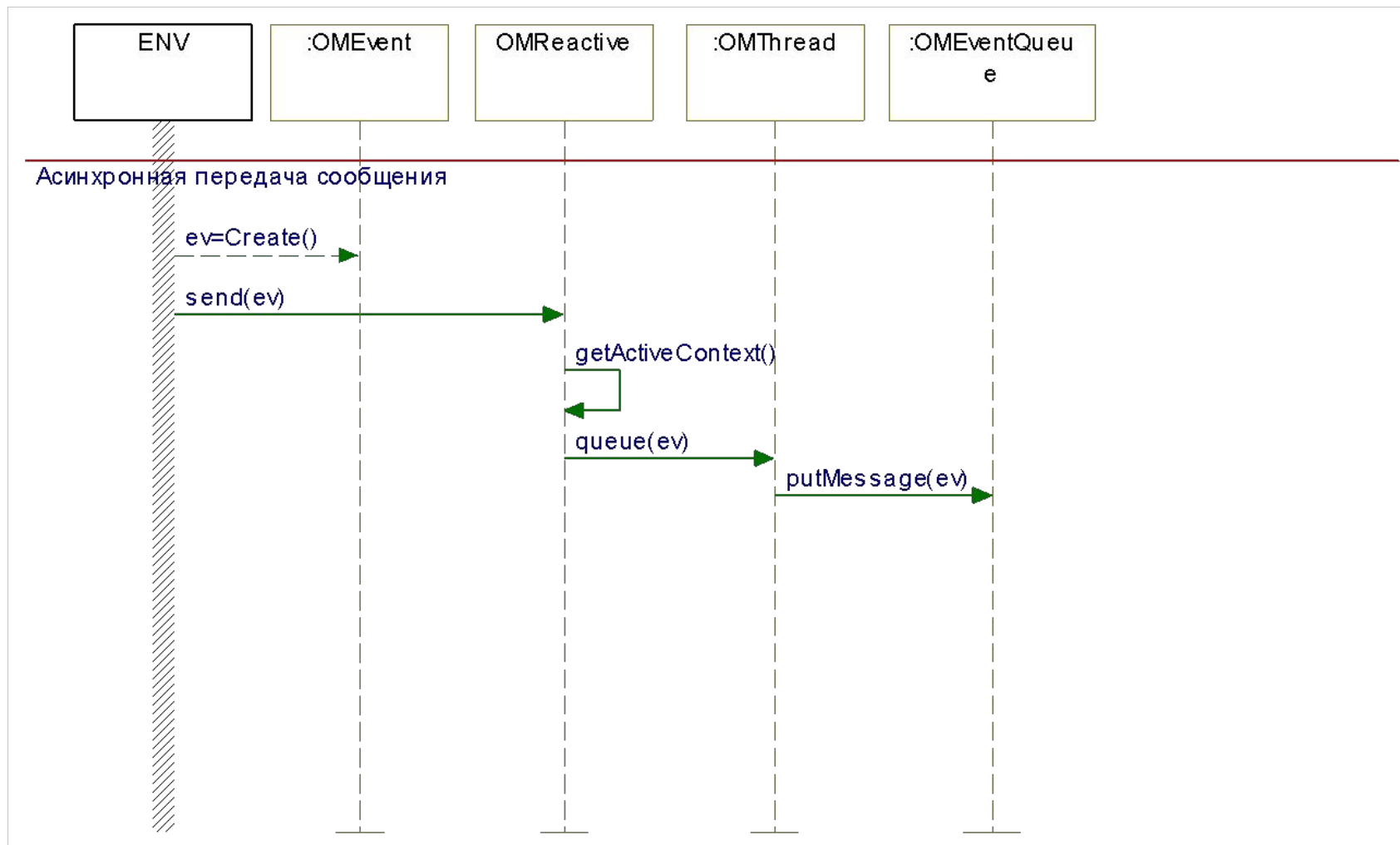


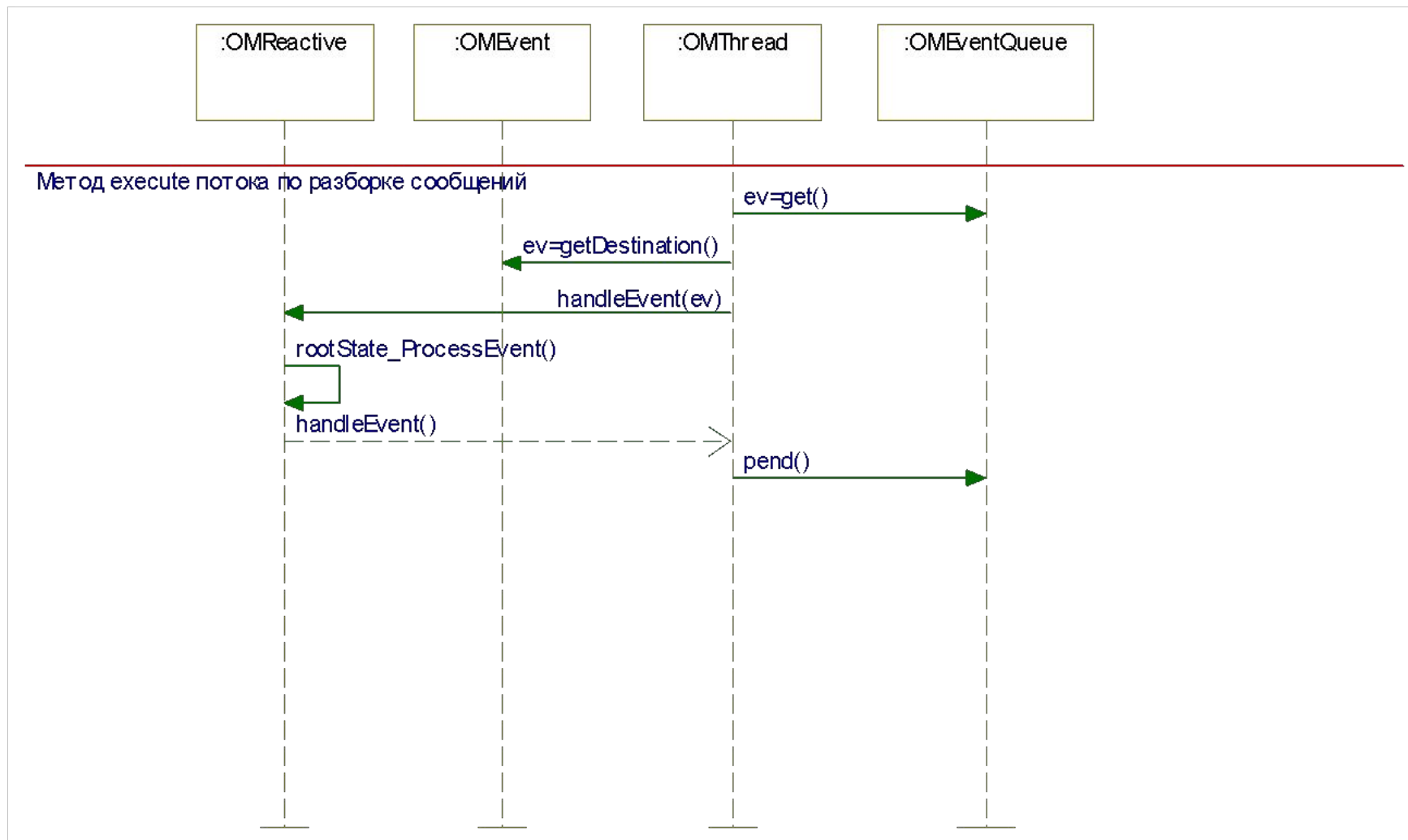
- **Асинхронные события**
- **События времени**
- **События вызова (синхронные)**

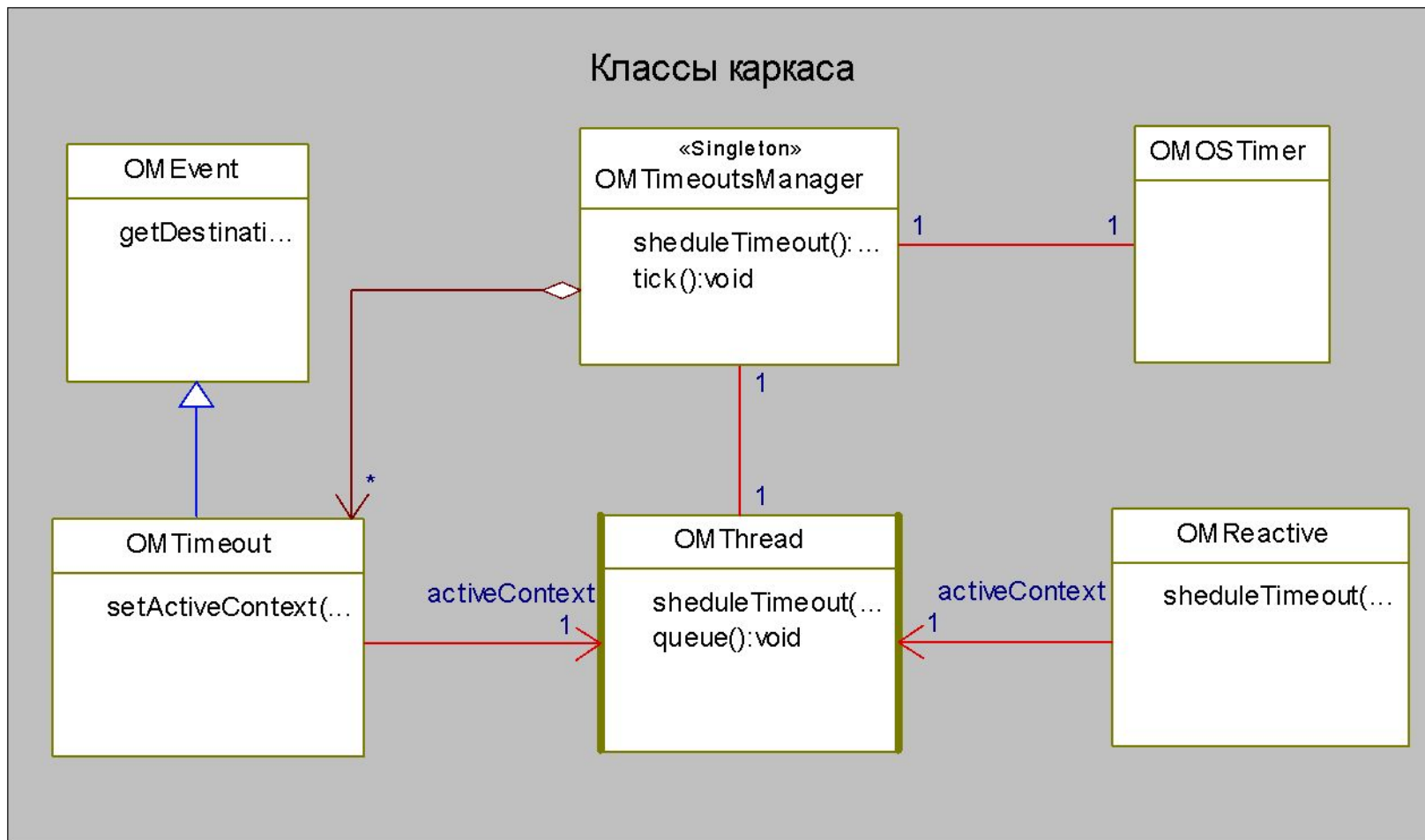


- Наследуется от класса OMThread каркаса
- Запускает в отдельном потоке функцию Execute
- Содержит очередь событий
- Предоставляет функцию queue для помещения событий в очередь
- В Execute разгребаёт очередь, передавая события адресатам на обработку в функцию handleEvent
- Позволяет перекрыть Execute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

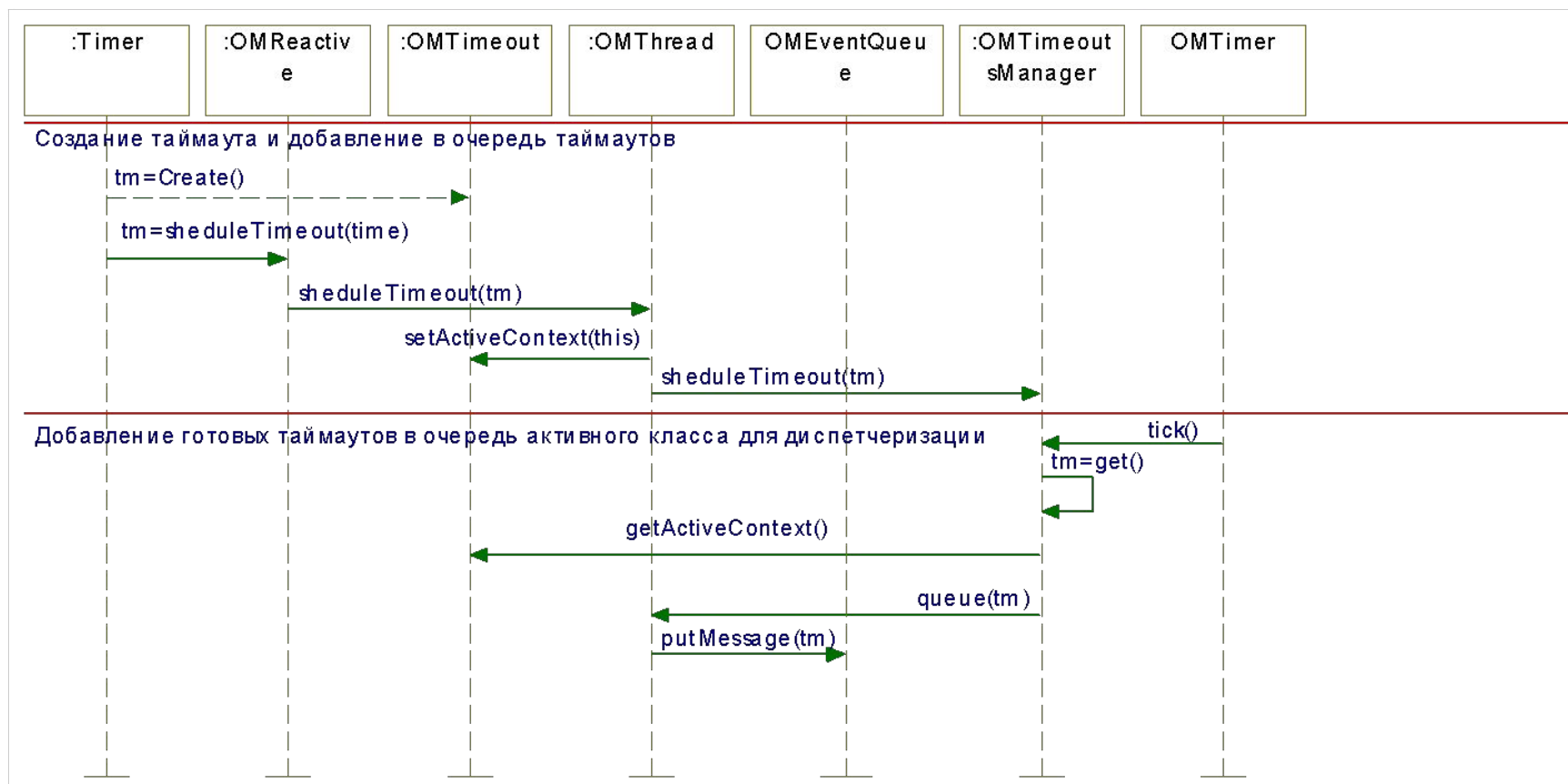
- Наследуется от класса OMReactive каркаса
- Предоставляет функцию send для передачи классу асинхронных событий
- Помещает полученные асинхронные события в связанный с ним активный класс для диспетчеризации
- Получает события от активного класса на обработку, вызывающего его функцию handleEvent
- Вызывает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



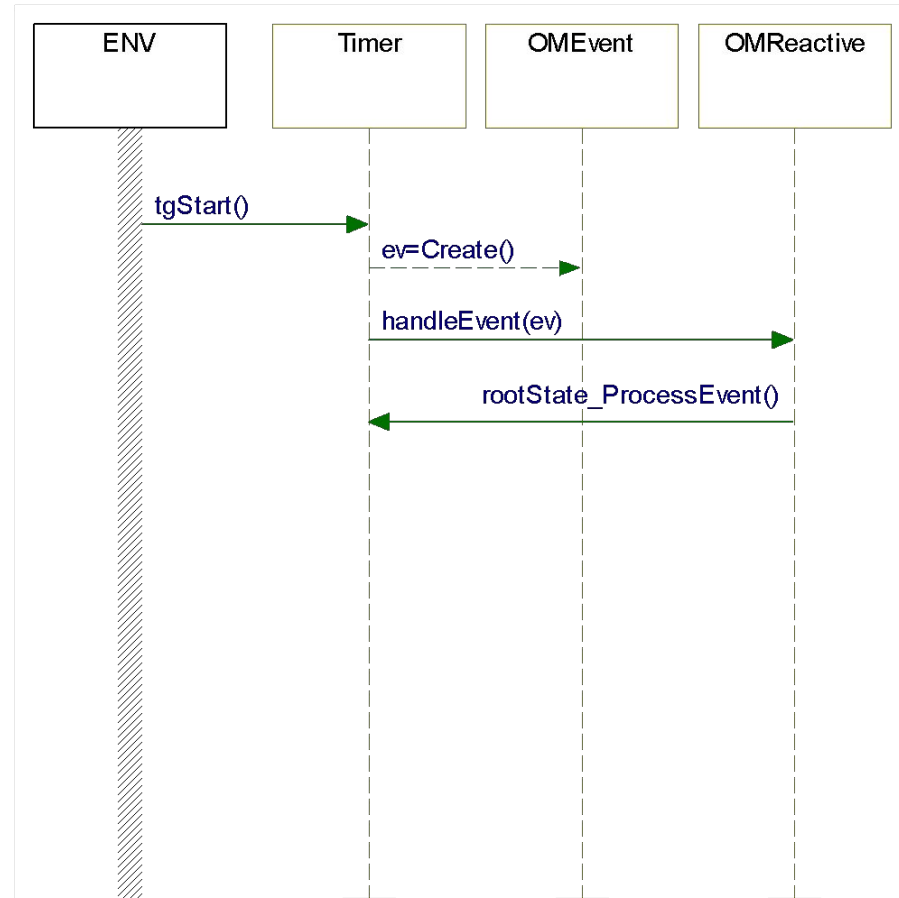


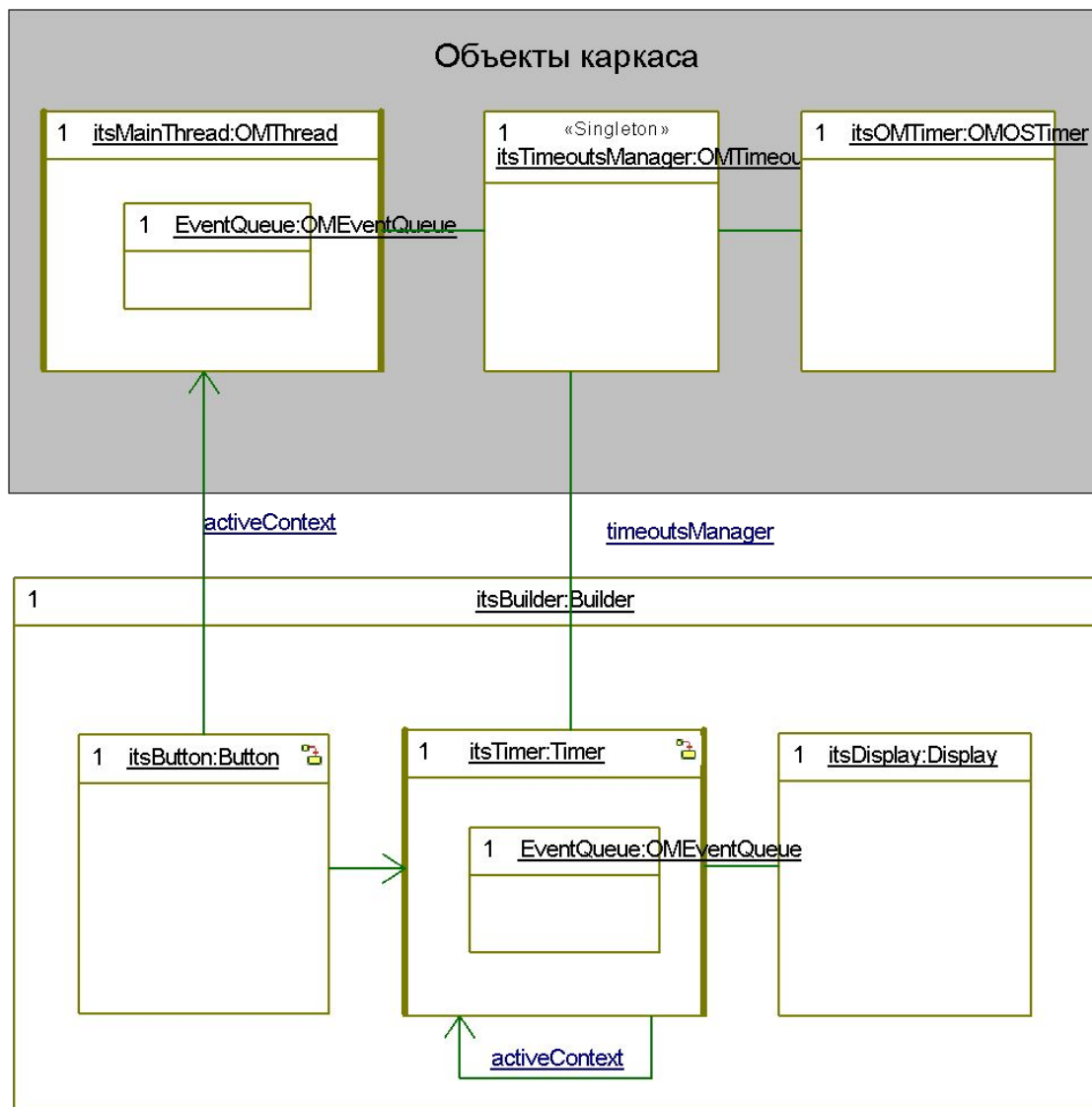


- Таймауты – это особый вид событий на которые можно определять реакции на диаграмме состояний
- Таймауты создаются в сгенерированном коде при входе в состояние и уничтожаются при выходе
- Всеми таймаутами управляет объект `TimeoutsManager`
- При истечении таймаута `TimeoutsManager` помещает его в очередь 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)**



<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