# VLT Software Workshop - April 1999

# Event Handler (EVH)

G.Chiozzi

22/04/1999

# Contents


# INTRODUCTION

**The Event Handling Toolkit is a set of C++ classes to develop event driven applications on workstations.**
**Development of applications requires a knowledge of concepts for event driven programs and object-oriented programming techniques.**

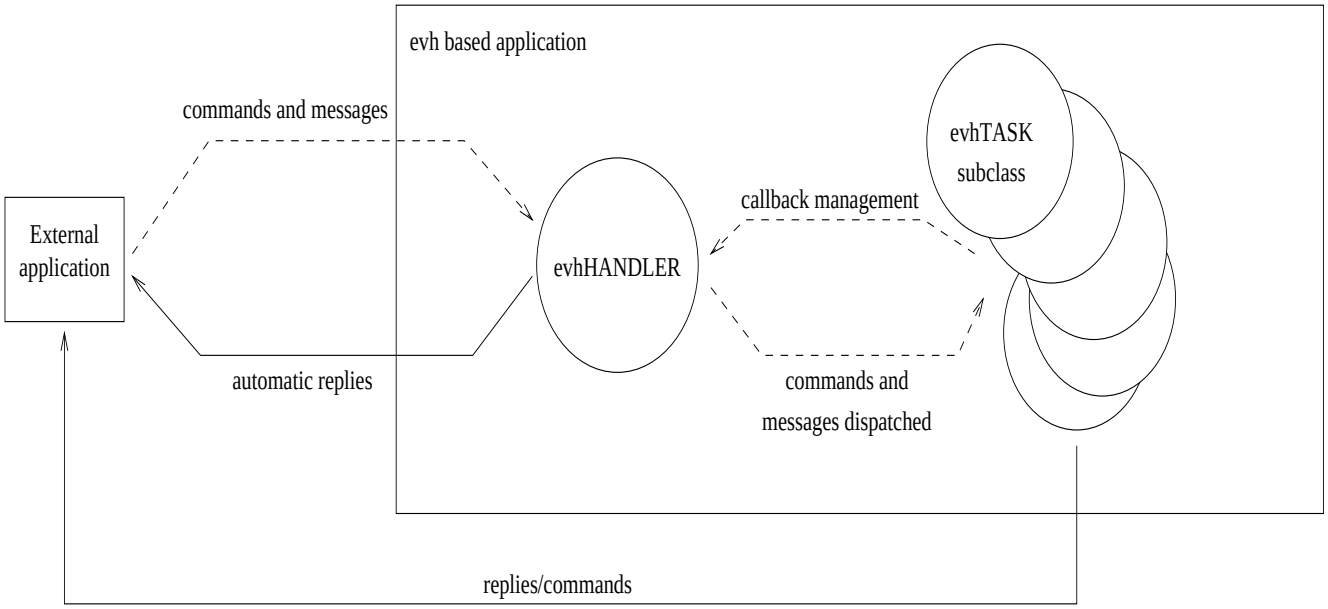
It runs only on Workstation.

Purpose of EVH is:

- Provide a framework to implement easily workstation applications that receive and send commands and messages.
- Implement in a standard way common application patterns
- Prevent common errors in applications
- Support object-oriented design and development
- Provide tools for testing and debugging of applications.

# The main loop

- Processes are designed in an event driven way
- The implementation is based around the *evhHANDLER* class

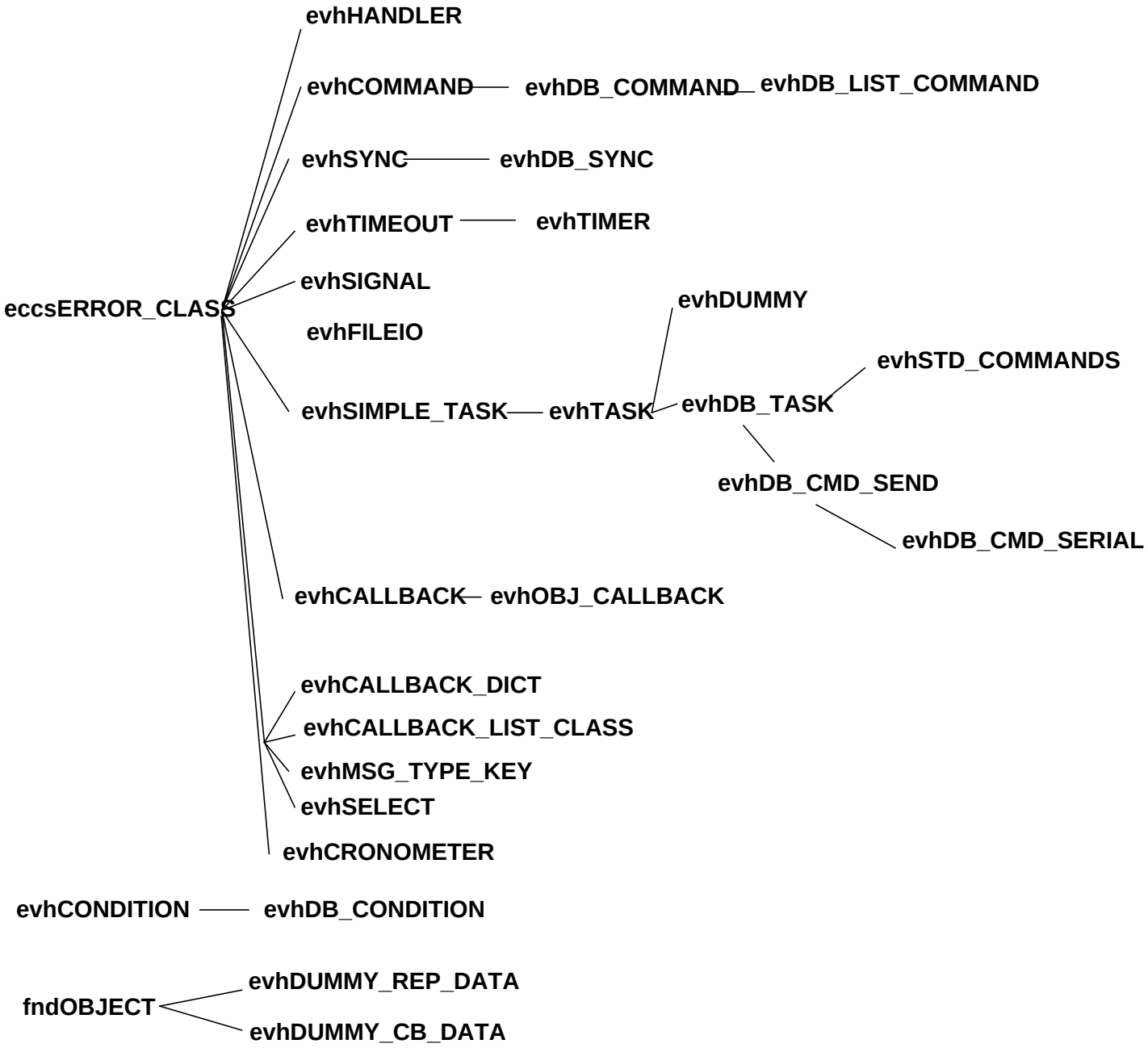



**The design of every process consists mainly in the design of the independent objects providing the methods to be attached as callbacks to the events**


# evh features

- The following events are handled:
  - commands
  - replies and error replies
  - time out notifications
  - cyclic timers
  - data base events
  - alarms
  - signals
  - file input
- callbacks are stored in a dynamic dictionary
- callbacks must return within a “command response time” of 1 second
- a wide set of ready-made classes implements standard behaviors and event handling protocols

# evh class hierarchy



# Using EVH

## In the Makefile:

- link the CCS standard libraries
- link the C++ standard libraries (`the vltMakefile define C++ should be used`)
- link the required support library `fnd`
- link the ECCS library `eccs`
- link the EVH library `evh`

```
Example:

EXECUTABLE=evhTest

evhTest_OBJECTS= eccsTest
evhTest_LIBS   = evh eccs fnd C++ CCS
```

## In the code:

```
#include "evhHANDLER.h"
int main(int, char *argv[])
{
    ccsCOMPL_STAT stat;
    stat = ccsInit(argv[0]);
    eccsErrLog(stat,"Error in Init ");

    evhHandler->MainLoop();

    stat = ccsExit();
    eccsErrLog(stat,"Error in ccsExit");
}
```

# The standard VLT application

**A standard VLT application can be implemented based on the evhSTD_COMMANDS, that provides:**

- Standard commands:
  - PING
  - INIT
  - STATE,STATUS
  - STANDBY,ONLINE,OFF
  - SELFTEST, TEST
  - SIMULAT, STOPSIM
  - VERBOSE
  - VERSION
  - STOP
  - EXIT, KILL, BREAK
- Standard database point:

```
CLASS evhSTD_COMMANDS
BEGIN
    ATTRIBUTE INT32 state evhSTATE_OFF
    ATTRIBUTE LOGICAL simulation
    ATTRIBUTE LOGICAL verbose
END
```

# The example application

**The following example is a typical WS coordination application base on the evhSTD_COMMANDS class.**

**It can:**

- Receive SETVAL and GETVAL commands. They are re-routed to another application, on LCU, for processing.
- Handle the TEST command. It is processed internally.
- Handle all VLT standard commands

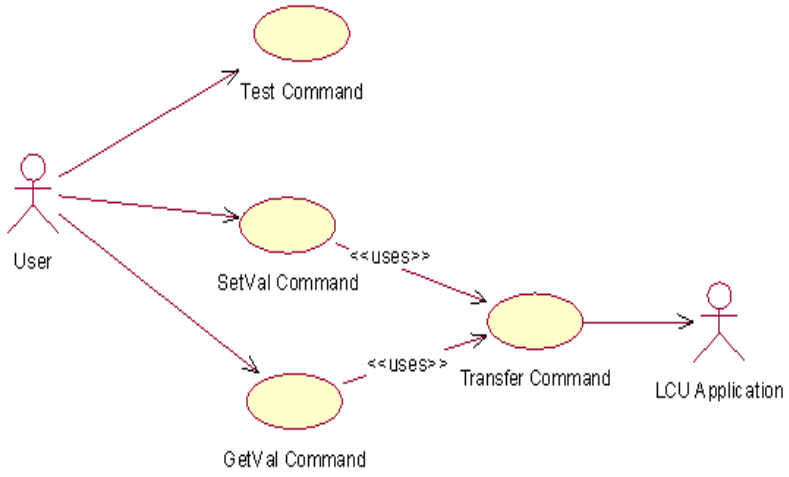
**It is a fully compliant VLT application and all commands are handled asynchronously.**

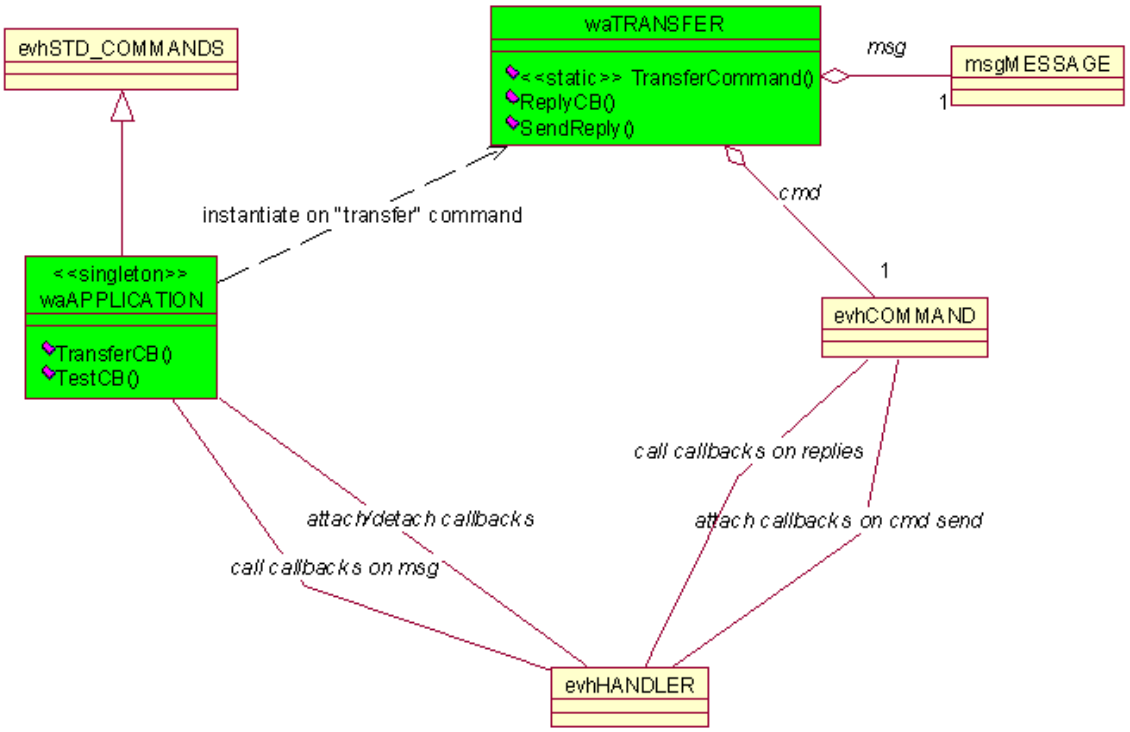
**In this session we show:**

- The structure of the application
- The typical design patterns used
- Testing and debugging using VLT development tools (in particular the usage of ccseiMsg and evhDummy for manual testing).

**Actual code is provided in appendix.**

# Example application architecture



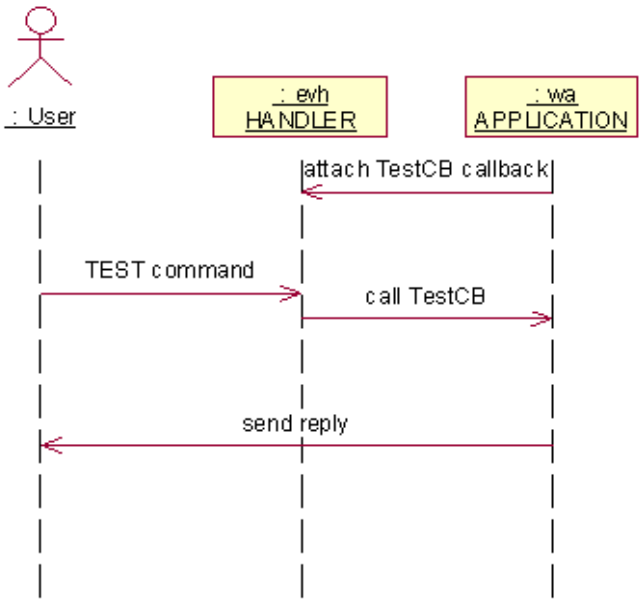



# Handling “immediate” commands

**“Immediade” commands are the simples case: an application receive a command, deals with it and sends an immediate reply to the originator.**

**“Immediate” commands are implemented writing yust one callback to receive the command.**

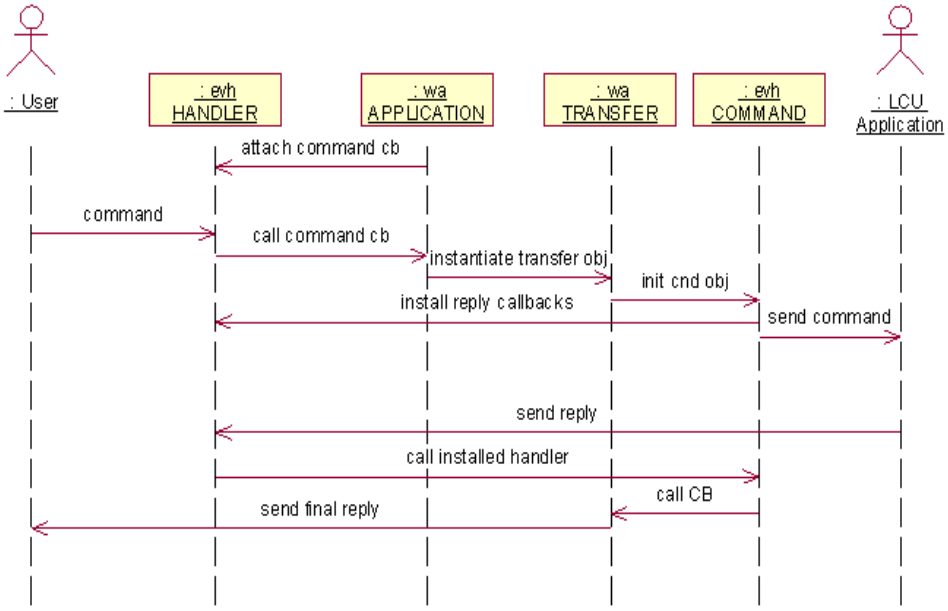
**In the example, TEST is an immediate command**



# Handling “transfer” commands

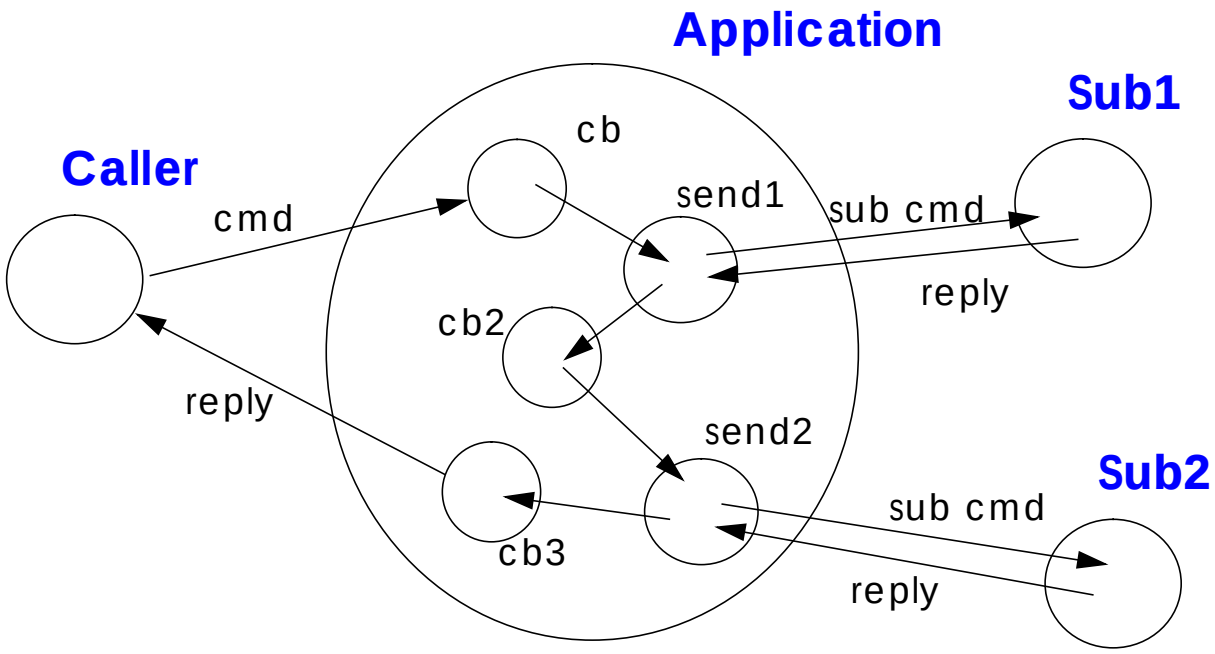
**“Transfer” commands are a typical case for workstation coordination applications: an application receives a command, deals with it and send one sub-command to the controlled processes. When they are done it must sends the final reply to the originator.**

**In the example, SETVAL and GETVAL are transfer commands.**



# Multi step commands

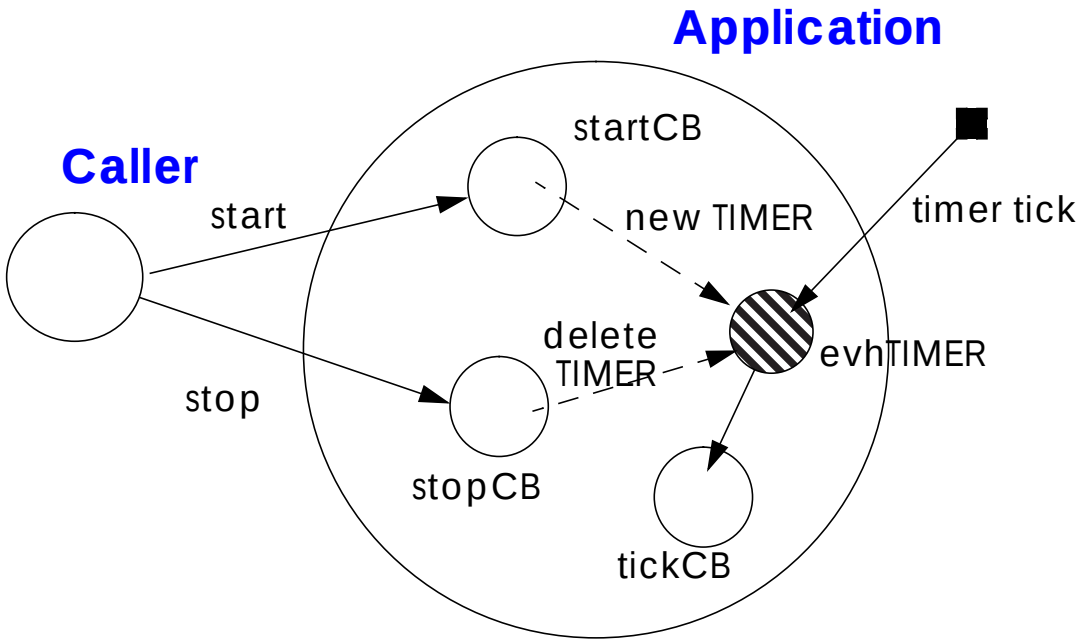
**In “multi step” an application receives a command, deals with it and send one sub-command a controlled processes. When the reply comes, some new elaboration is done and a new command is sent and so on for as many steps are needed.**



# Cyclical timers

**Another typical design pattern involves starting and stopping (for example via commands) a cyclical timer.**

**"Cyclical timers" are implemented using the evhTIMER class.**




# Conclusion

This presentation has shown only some of the most important aspects of the library.

For more details look at:

- VLT Software - CCS Event Tool Kit User Manual (VLT-MAN-ESO-17210-0771)
- Guidelines for the Development of VLT Application Software (VLT-MAN-ESO-17210-0667)
- Code examples in evh <mod>/test directory