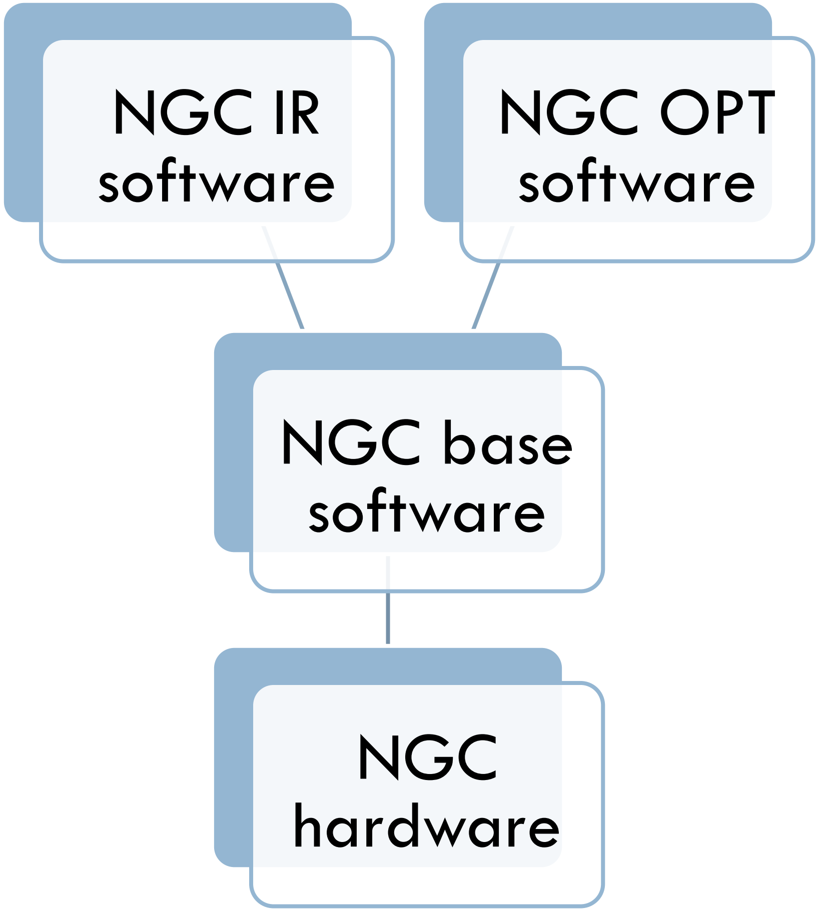
INNOVATIVE SOFTWARE ENGINEERING APPLIED TO NGC FOR OPTICAL DETECTORS

Claudio Cumani – Andrea Balestra

INS Monthly Meeting - 2007, October 26

Software for the NGC controller

- NGC = New General detector Controller
  - Basically same electronics for both infrared and optical detectors
- NGC software
  - Base software, to interface to the NGC hardware (common to infrared and optical detectors)
  - Control software (different for infrared and optical detectors)
- **Why 2 different flavors of software for NGC?**



Differences btw IR and OPT detector controllers: intrinsic

- **"Exposure" handling**
  - **Optical**

    Rigid scheme for exposures (*wipe* - *integrate* - *read*).

    Active intervention of the control-server during the exposure is required (application of new voltages in each state).

    "Active" interface to different kinds of shutter controllers (open/close, status check, open/close delays, etc.).
  - **Infrared**

    Detector continuously read-out (infinite loop).

    Starting an exposure = starting transfer and storage of data. Once exposure is started, control server mainly reacts passively on incoming data-frames.

    No "active" interface to external devices (interfaces through trigger signals, e.g., for *nodding*).

Differences btw IR and OPT detector controllers: intrinsic

- **Data handling**

  - **Infrared**

    Computationally intensive different data pre-processing, read-out mode dependent.

  - **Optical**

    Detector read-out just once at the end of an exposure.
    The only processing to be done is pixel sorting and offset calibration (centroiding and bias-subtraction on request).

Differences btw IR and OPT detector controllers: historical

- IRACE is usually used as a "black box". For special acquisition purposes, it offers a set of configurable "building blocks", code classes, etc., from which instrument builder can develop what they need;
- FIERA has been requested to be always a "black box", which hides all the differences between system configurations (same code must cope with all possible requirements/configurations);
- In addition optical detector controllers are requested to interface/control also devices which are not – strictly speaking - part of the detector, like vacuum and temperature control (and write values in FITS file header)

“Good” example of “bad” requirements

- NGC should be backward compatible with previous IRACE and FIERA controllers
  - i.e.: infrared and optical NGC should keep the IRACE and FIERA – different – interfaces
- Optical and infrared NGCs should look the same
  - i.e.: infrared and optical NGC should have the same interface

→ Compromise needed

Optical NGC needs its own software

- Impossibility to reuse the FIERA software: NGC hardware/software architecture is “IRACE-oriented” (no DSP, setup-driven replaces database-driven configuration, etc.)
- Impossibility to use IR NGC control software (IR/OPT differences)
- Base software replaces the FIERA DSP code for interfacing with hardware (thanks Joerg), but need to develop the rest.
- Learning from experience:
  - Produce a sw package where structural modifications are easy
  - Take advantage of existing software packages, as well as of tools for automatic code generation

Optical NGC needs its own software

In practice:

- avoid to “reinvent the wheel”

D
Zahn
"I CALL IT A 'WHEEL', BUT I DON'T HAVE ALL THE BUGS OUT OF IT YET."

- when possible, let someone else do the dirty job (message/database/error handling, etc.)!

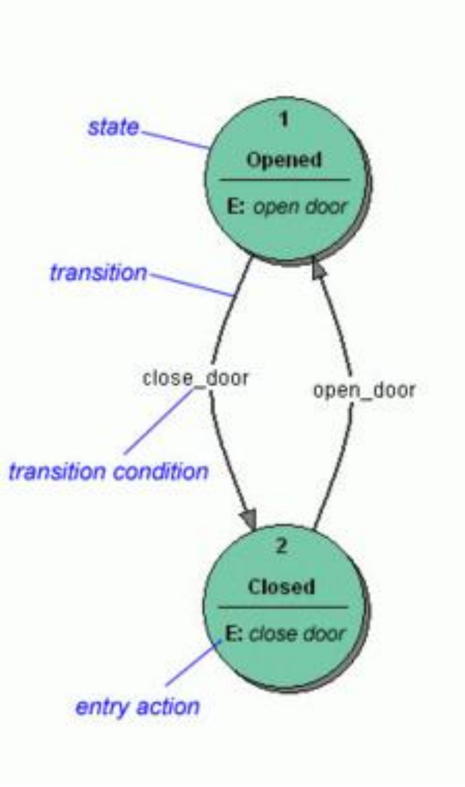
NGCOSW development – state machine concept

Detector controllers can be modeled as finite state machines

- **finite state machine**

  model of behavior composed of a finite number of states, transitions between those states, and actions.

  - Powerful ability to implement decision making algorithms
  - Easy to create (table of possible states and relations among them)
  - The design process involved in creating a State Machine improves the overall design of the application.
  - **Restructuring is very easy**
  - **Only model that allows "easy" code generation (for state transitions)**



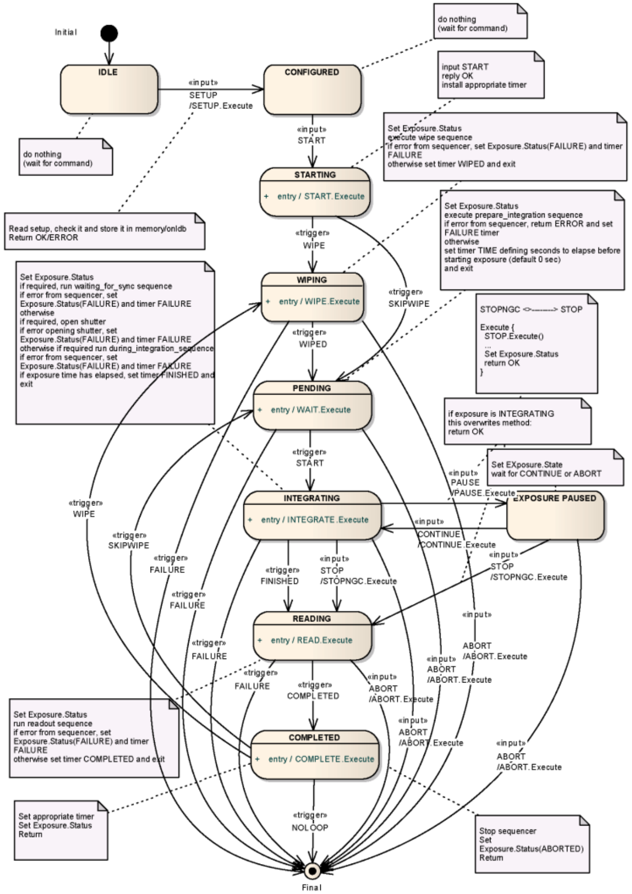
NGCOSW development – state machine concept

Exposure state machine

(just an example: don’t try to read it ☺)

Designed using UML (Unified Modeling Language) with Enterprise Architect.

From this model, code can be automatically generated!



do nothing
(wait for command)

do nothing
(wait for command)

input START
reply OK
install appropriate timer

Read setup, check it and store it in memory/onldb
Return OK/ERROR

Set Exposure.Status
execute wipe sequence
if error from sequencer, set Exposure.Status(FAILURE) and timer FAILURE
otherwise set timer WIPED and exit

Set Exposure.Status
execute prepare_integration sequence
if error from sequencer, return ERROR and set FAILURE timer
otherwise
set timer TIME defining seconds to elapse before starting exposure (default 0 sec)
and exit

Set Exposure.Status
if required, run waiting_for_sync sequence
if error from sequencer, set Exposure.Status(FAILURE) and timer FAILURE
otherwise
if required, open shutter
if error opening shutter, set Exposure.Status(FAILURE) and timer FAILURE
otherwise if required run during_integration_sequence
if error from sequencer, set Exposure.Status(FAILURE) and timer FAILURE
if exposure time has elapsed, set timer FINISHED and exit

```
STOPNGC <>---------> STOP

Execute {
  STOP.Execute()
  ...
  Set Exposure.Status
  return OK
}
```

if exposure is INTEGRATING
this overwrites method:
return OK

Set EXposure.State
wait for CONTINUE or ABORT

Set Exposure.Status
run readout sequence
if error from sequencer, set Exposure.Status(FAILURE) and timer FAILURE
otherwise set timer COMPLETED and exit

Set appropriate timer
Set Exposure.Status
Return

Stop sequencer
Set
Exposure.Status(ABORTED)
Return

NGCOSW development – automatic code generation

Code generation tool: wsf (workstation software framework)

by Luigi Andolfato (SDD), used – for instance – for TCS and APE

1. automatic code generation from state design (described by a configuration file) (*)

   "automatically generated" code handles state transitions, messages, commands, error conditions, etc.
   (NOT the actions needed to drive an exposure!)

2. implementation of detector control code (CCD, shutter, etc)
3. feedback to SDD

(*) Library created to generate wsf configuration files from Enterprise Architect UML state diagrams

NGCOSW development – integration in VLTSW

Usage and integration of other VLTSW tools:

- configuration: ctoo, stoo
- FITS keywords handling: oslx
- image data transfer: dxf
- command dispatching (base for the *super-DCS* process): cdp

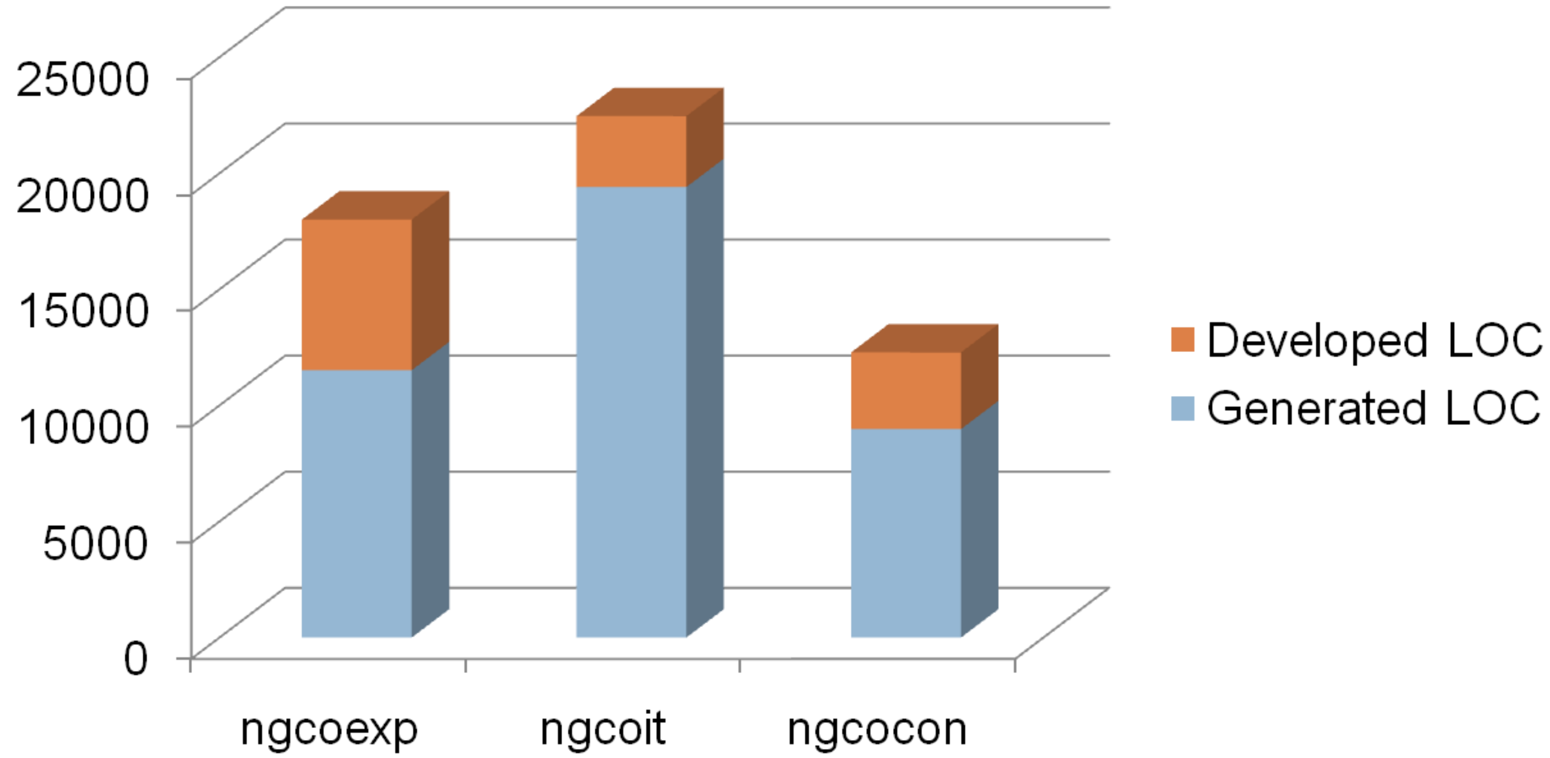
NGCOSW development – statistics

Rough estimate (on last archived version, 3.20)

LOC = Lines Of Code (trying to describe the work in a quantitative way…)

- exposure coordination process: total LOC: 18006
  developed LOC and configuration: 6488 (36%)
- image transfer processes: total LOC: 22462
  developed LOC: 3055 (13%)
- coordination control process: total LOC: 12283
  developed process LOC: 8 (0%) (*adapted cdp*)
  developed LOC (utilities, scripts, oldb configuration): 3304 (26%)

NGCOSW development – statistics



NGCOSW development – statistics

For the last archived NGCOSW version (3.20):

NGCOSW = 64004 LOC, 14282 developed (22%)

NGCOSW development – some comments on statistics

- “*Measuring programming progress by lines of code is like measuring aircraft building progress by weight*” (Bill Gates)
- “Rough” estimate: includes code, database configuration, system configuration, utilities, etc.
- The code originally written for NGCOSW was more, but code has been moved into general purpose VLTSW packages

NGCOSW development – where do we stand

- NGCOSW August 2007 release
  - CCD mosaics readout from one or more outputs
  - Data displayed on RTD and saved in FITS files with “one extension per chip” format
  - Public command and online database interface
  - User Manual

  Delivered to MUSE consortium

  Used for MUSE and Zimpol prototypes

  Integrated in VLT Control Model (BOSS)

- NGCOSW December 2007 release will have
  - Interface to the new NGC shutter module
  - GUI
  - Telemetry (temperature and vacuum control and monitoring)

NGCOSW development – pro and cons

- **Time**

  Additional time spent by being NGCOSW the first user of wsf and cdp packages outside the SDD division: interactions with the package developer (Luigi Andolfato) to have support and addition of functionalities needed by NGC.

  But: time saved from coding moved to testing and optimization.

- **Standardization**
- **Integration**

  NGCOSW triggered the integration of dxf (data transfer facility), signal handling and file I/O within wsf. Some utility developed by ODT has become part of the wsf module.

  NGCOSW also triggered the improvement of FITS extension handling within oslx.

- **Code robustness, flexibility and maintainability**

  Test-driven development (design → system test of generated code → coding) → robustness

  State machine = better structure, easy to change → flexibility, maintainability

  The part of the code which was explicitly developed to control optical exposures is minimal and well confined, i.e., it is easier to implement new features (and "less developed lines = less bugs!)

- **Dependence on other software packages**

The end

The end?

Coming soon: SysML and systems engineering triggered by our experience with DOORS, requirement handling, design tools, etc.…

… matter for a next talk!