Instrument Control Software Requirement Specification for Extremely Large Telescopes

Peter J. Young[*a], Mario J. Kiekebusch[b], Gianluca Chiozzi[b]
[a]Research School of Astronomy and Astrophysics, The Australian National University, Cotter Road, Weston Creek, ACT 2611, Australia;
[b]European Southern Observatory, Karl-Schwarzschild-Strasse 2, D-85748 Garching bei München, Germany

ABSTRACT

Engineers in several observatories are now designing the next generation of optical telescopes, the Extremely Large Telescopes (ELT). These are very complex machines that will host sophisticated astronomical instruments to be used for a wide range of scientific studies.

In order to carry out scientific observations, a software infrastructure is required to orchestrate the control of the multiple subsystems and functions. This paper will focus on describing the considerations, strategies and main issues related to the definition and analysis of the software requirements for the ELT's Instrument Control System using modern development processes and modelling tools like SysML.

Keywords: ELT, software requirements, instrument control system, framework

1. INTRODUCTION

This paper describes work underway at ESO on the development of a software infrastructure for the planned European ELT (E-ELT). There has been much discussion in the telescope software domain (both inside ESO and at other institutions) about the usefulness of software infrastructures developed for the 8m-telescope era. Many of these discussions have focused on technologies used, processes adopted, software development and maintenance costs and strategies followed. In other words, there has been much introspection.

In addition, new 'impressive' technologies have been seized upon in a rather 'bottom-up' fashion with the hope that they can provide magical solutions to some of the unsatisfactory outcomes of current software systems. In particular, development and maintenance costs of existing systems are considered to be too high, with the amount of in-house development undertaken often blamed as a significant contributing factor. Inflexibility of developed systems has also been thought to be a major shortcoming with attempts at extending functionality or integrating new generation capability proving to be costly and difficult. Hence, much focus has been directed toward determining how to reduce in-house development and build in flexibility.

With these considerations in mind ESO have started work on the control software for the E-ELT. The E-ELT is planned to have up to ten complex scientific instruments (including two post-focal adaptive optics systems) that will be built by external consortia (usually made up of institutions located in ESO member countries). The E-ELT TCS is planned to have many of its subsystems built externally under sub-contract. For all of these endeavours it is considered that there is much advantage to be gained by ensuring a level of commonality in operation software is enforced by design. ESO will need to develop the infrastructure that achieves this. To achieve an unbiased view, ESO have decided to involve external collaborators and contractors in the process undertaken for analyzing and building this infrastructure and described in this paper. For this reason one of the authors (Young) has been working under contract at ESO, providing experience from Gemini and other projects.

[*] pjy@mso.anu.edu.au; phone 61 2 61250281; fax 61 2 61250233; anu.edu.au

2. SOFTWARE FRAMEWORKS

A software framework as discussed in this paper refers specifically to both the E-ELT Instrument Software Framework (INS-FW) and the E-ELT TCS Core Infrastructure Framework (CIF). More generally, a software framework is "a re-usable design for a software system (or subsystem). A software framework may include support programs, code libraries, a scripting language, or other software to help develop and glue together the different components of a software project. Various parts of the framework may be exposed through an API"[1]

Examples of software frameworks used previously in the telescope software domain include the VLT Software (VLT SW), the Gemini Core Instrument Control System (Gemini CICS) and the ALMA Common Software (ACS).

2.1 Measuring effectiveness of software frameworks

In order to evaluate the potential usefulness of an INS-FW, ESO considered it important to determine the positive impact (if any) of the current VLT CCS on the development of first and second-generation instruments for the VLT. This would provide a basis for determining the scope of any common software framework for the E-ELT project.

Comparing development costs of different instruments is not easy. Development effort, developer skill and instrument complexity all need to be taken into account. Subsequent maintenance effort also needs considering. ESO have started to work on such an analysis, but until now no conclusive estimates have been produced, because of the inherent difficulties mentioned above and of the fact that neither complexity nor developer skill can easily be measured.

For the time being, ESO have, instead, relied on their experience of instruments operating on the VLT both with and without infrastructure software to be confident that the development of a suitable infrastructure for the E-ELT is important, noting the following:

- Frameworks offer the potential for maximizing software reuse. They also afford the opportunity for applying a well-understood design pattern to common software in a consistent fashion. Not using a framework could mean that common problems for different instruments or sub-systems are solved with different efficiencies, have potentially costly errors (e.g., leading to a system crash) and require different developers to 'reinvent the wheel'.
- On the other hand – using a framework could mean: that a developer has to learn about services that are not needed; innovation can be stifled; code bloat can occur (support for specific systems provided in a generic way can be expensive)

Looking at the other example framework cited earlier (i.e., the Gemini CICS), Gemini have found that development of the instrument control system for many of its instruments by instrument developer groups at institutions located in partner countries has not gone as easily and smoothly as originally planned. As stated by Gillies et al (2008), first and second generation instrument software development were affected by delays and cost overruns with sometimes a factor of two overspent. Others were delivered with defects requiring significant effort for Gemini to correct. Instrument developers complained that they wanted fewer constraints from Gemini so as to utilise their own software and reduce costs.

Gemini noted the obvious advantage of using framework software when developing systems but also took note of developers who often have difficulty using such frameworks – especially up front learning curve issues. Gillies et al state that for their Aspen instruments, Gemini has focused on creating a minimum amount of software needed by builders to integrate with the Gemini environment – providing more freedom and choice for the developer. Flexibility and reliability have been considerations from the beginning. This is a move away from a deeply involving framework, such as the CICS, and code reuse in general – which hasn't been a success for them (Gillies et al, 2008). They have found that most instruments are sufficiently unique that their software still requires a Gemini developer (to be assigned to the instrument) who is a specialist dedicated to learning the intricacies of the instrument's software.

Both the VLT and Gemini experiences (and others) are informing the development of software frameworks for the ELTs, i.e., frameworks are important but so are framework flexibility and the maximum utilisation of developer experience. These aspects need to be incorporated in any new software infrastructure.

[1] http://en.wikipedia.org/wiki/Framework

3. FRAMEWORKS FOR THE ELT ERA - REQUIREMENTS ANALYSIS

The E-ELT Instrument Control Software Framework (INS-FW), combined with specific extensions to support each instrument, includes all the software components and communication infrastructure required for Observatory level control of each instrument's configuration, control and data acquisition. The INS-FW will provide the mechanism for the coordination of scientific observations under each of the E-ELT's different instrument operating modes.

All E-ELT instruments currently being studied are unique, though all have some common functional attributes involving configuration, coordination and operation that provide for the possibility of a level of standardisation. From the E-ELT observatory point of view, a software framework should address these commonalities in a general way so that instruments are handled in a consistent, efficient and also flexible manner.

The INS-FW will provide a facility that both ESO development, maintenance and operations staff and external instrument developers can use to maximise the efficient development and operation of E-ELT instruments. It will provide the operational interface between the Observatory and the instrument but will not prevent the instrument from operating in stand-alone mode when necessary.

Also recognising the complexity and individuality of each of the E-ELT instruments, the INS-FW will not attempt to provide a complete generic facility for operating each instrument, as this will mean a resulting system that will be overly complex and inflexible. Instead, the INS-FW will provide the minimal set of functionality that assists developers in integrating their instrument for operation in the E-ELT environment (a strategy also adopted by Gemini – note earlier discussion)

It is assumed that ultimate responsibility for maintaining software for instrumentation and TCS sub-systems will reside with ESO after the development of this software by instrument consortia and sub-contractors, hence guidelines and tools for building this software to required standards will be provided as part of the supplied frameworks.

3.1 Developing the vision and scope for the framework

The first step in any requirements analysis phase is to define the scope of the software to be built. This is developed from the basic understanding of what is to be provided and is informed by previous experience.

The highest level goal is to:

- Provide a software infrastructure that can be used to configure, control and acquire data from E-ELT instruments

It is envisioned that this will be done by:

- A set of generic reusable building blocks (components) for controlling the individual instrument elements
- A graphical tool for developers to piece together these components to form a system in an arbitrary but controlled structure and that can be deployed in a distributed fashion
- An interface and communication infrastructure necessary for binding components of the system together
- A mechanism for the configuration of components
- A facility for extending a configuration for the integration of 'specialist' components built to support unique instrument characteristics
- A mechanism for the independent command and control of individual components within a system
- A mechanism for the interoperation of components and sequencing of operations between different components in the system
- A mechanism for the setting, inspection, monitoring and recording of individual and collective data items (attributes) belonging to components of the system
- A mechanism for the display of image data from both scientific and wave front sensor detectors
- A mechanism for the storage of both scientific and wave front sensor image data using the ESO E-ELT Data Products Specification which incorporates the FITS standard
- A mechanism for the logging of data item values and system events for eventual storage in a data base management system

- A mechanism to handle exceptional system events including the notification, acknowledgment, logging and clearing of alarms
- A mechanism for handling system errors and exceptions
- A set of tools for real-time monitoring of telemetry data
- A mechanism to interface with the Telescope Control System
- A mechanism to interface with the Observatory Astronomical Site Monitor to allow instruments access to prevailing astronomical and meteorological conditions at the site

3.2 High level requirements - business rules and system context

It is normal software-engineering practice for any business rules applying to a system be identified and stated at the outset. It is also important that all requirements stem from the product vision and scope and are constrained by these business rules. Business rules are derived from the specifics of the business task that the system (the INS-FW) is supposed to address. These rules have been sourced from the E-ELT top-level requirements documents.

When is a "requirement" a business rule and when is it just a requirement? Business rules can be classified as either "Facts", "Constraints", "Action Enablers", "Inferences" or "Computations" (Wiegers 2003 pp 154).

Some example E-ELT software business rules that have been identified include the following – space does not permit any further detailing. Suffice it to say that this step of understanding the business constraints is an important step in eliciting requirements for the system:

BR.1 The INS-FW shall comply with ESO safety policy [Constraint]

BR.2 The INS-FW shall comply with ESO product assurance and configuration control policies [Constraint]

BR.3 The software packages developed by instrument building consortia shall conform to software management, configuration and test procedures to be provided by ESO. ESO to provide said procedures in sufficient time [Action Enabler]

3.3 Determining the product breakdown

Table 1 shows the proposed preliminary product breakdown for the INS-FW. These separate software products combine together to form the INS-FW and have been specified as sub-systems that meet the high-level functional goals of the framework as specified in the previous section.

Table 1. Instrument Software Framework Product Breakdown.

	INS-FW Product Name	INS-FW Product Description
INS-FW 1.	INS-FW Component Library	Library of reusable software components
INS-FW 2.	INS-FW Component Tool	A tool for constructing a new software component for an instrument
INS-FW 3.	INS-FW Configuration Manager	System design and configuration tool responsible for building valid instrument configurations (including associated Configuration Database)
INS-FW 4.	INS-FW Supervisor	Responsible for instantiating and deploying a valid instrument configuration
INS-FW 5.	INS-FW Sequencer	Responsible for orchestrating instrument components to perform a required action
INS-FW 6.	INS-FW Parameter Database	Active distributed database of parameters/attributes of components (may be shared with CIF)
INS-FW 7.	INS-FW Alarm Handler	Common alarms handling system (may be shared with CIF)
INS-FW 8.	INS-FW Error Handler	Error and exception handling system (may be shared with CIF)
INS-FW 9.	INS-FW Logger	Common logging system (may be shared with CIF)
INS-FW 10.	INS-FW GUI Generator	GUI Generator for constructing and automatically generating graphical user interfaces
INS-FW 11.	INS-FW Data Manager	Image data manager responsible for gathering final image and header data and formulating the data product
INS-FW 12.	INS-FW Display Manager	Image display system
INS-FW 13.	INS-FW Telemetry Manager	Tools for monitoring telemetry data

A context diagram (Figure 1) shows how the INS-FW fits within the E-ELT environment. External sub-systems identified include the Telescope Control System (TCS), Adaptive Optics System (AOSystem), Time Reference System

(TRS), Observatory Data Archive (ODA), Observatory Control System (OCS), Interlock System (ILS) and Astronomical Site Monitor (ASM). Interfaces (both internal and external) will have to be developed for each of the systems identified.

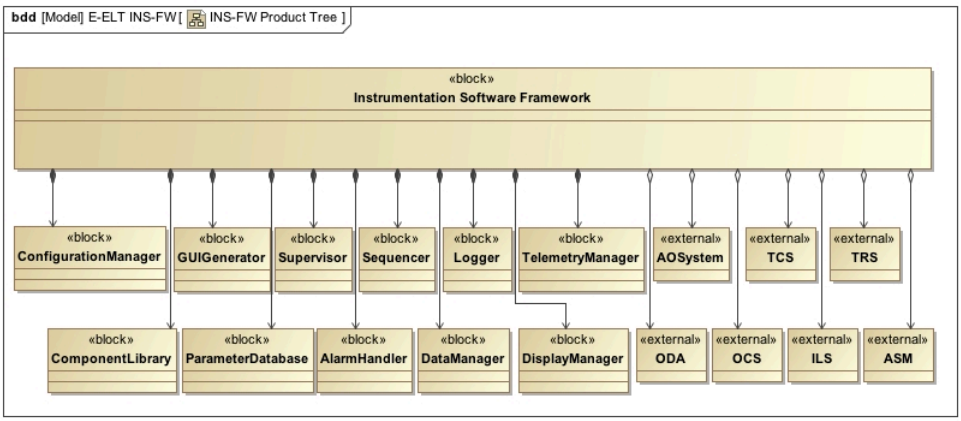

Figure 1. INS-FW Product Tree and referenced E-ELT Sub-Systems

3.4 Use-Case analysis - extracting the functional requirements

High-level system scenarios envisioned for the INS-FW have been developed. These scenarios describe how the system interacts with the identified users and the external E-ELT systems. They provide the basis for specifying system behavioural requirements. A complete set of scenarios, corresponding to each primary and alternative path for each use case, are needed to completely specify the requirements.

The use cases described here have been derived keeping in mind the following: "*Important use-cases are those in which the user derives the most benefit. To build a software architecture one needs the smallest set of use cases necessary to cover the important aspects of the system*" (Armour & Miller 2001).

Primary scenarios identified for the INS-SW include:

- Infrastructure (engineering) use cases:
  - Constructing an instrument component and adding it to the INS-FW Component Library
  - Building an instrument configuration from INS-FW components
  - Building/generating instrument operation and engineering GUIs
  - Starting the INS-FW to support an instrument configuration
  - Shutting down the INS-FW
  - Directly access/control an instrument component
- Observation use cases
  - Set the instrument operating mode
  - Selecting an available observation block
  - Readying (setting up) an instrument for an observation
  - Acquire the scientific object of interest
  - Taking an observation
  - Interrupting an observation (aborting, ending or pausing)

Some of these primary scenarios can be further specialised, for example the *Taking an Observation* scenario can be further detailed into particular observation categories – modelled using the UML generalisation relationship:

- Taking a simple CCD observation
- Taking an infra-red observation

In addition, to illustrate the requirement to support more complex scenarios, the *Simple CCD Observation* can be extended to include more complex observation types likely to need supporting. We cannot hope to detail all possible extensions here, but the two examples presented should indicate the basic *extensibility* requirement of the framework when dealing with observations, modelled using the UML extension relationship:

- Taking nod & shuffle exposure (typically used to minimize sky background noise for faint objects)
- Taking a compound observation (e.g., an un-equal length exposure for an instrument with more than one camera)

Each of these scenarios has been described in more detail using UML use-case diagrams and descriptions. For the purpose of this paper, we summarise them in the following Table 2 and include an example use case diagram (Figure 2).

Table 2. Summary of Use Cases Developed for the INS-FW

Use Case	Primary Actor	Secondary Actors	Use Case Type
Build Instrument Configuration	Configuration Manager	Configuration Database	Primary
Build Instrument GUI	Developer	Configuration Database	Primary
Construct Component	Developer	Configuration Database	Primary
Start INS-FW	Operator	Configuration Database, TCS, Instrument	Primary
Set Instrument Mode	Operator	Instrument	Primary
Directly Access and Control Component	Engineer	Instrument	Primary
Shutdown INS-FW	Operator	Instrument, TCS	Primary
Select Next Scheduled Observation	Operator	High Level Operation System	Primary
Acquire Object	Operator	Instrument, TCS	Primary
Ready Instrument for Observation	Operator	Instrument	
Take Observation	Operator	TCS, Instrument, Display, Data Archive	Abstract
Take a CCD Observation	Operator	TCS, Instrument, Display, Data Archive	Derived from Abstract Observation
Tan Infra-red Observation	Operator	TCS, Instrument, Display, Data Archive	Derived from Abstract Observation
Taka a Nod & Shuffle Observation	Operator	TCS, Instrument, Display, Data Archive	Derived from CCD Observation
Take a Compound Observation	Operator	TCS, Instrument, Display, Data Archive	Extended from CCD Observation
Interrupt Observation	Operator	Instrument	Abstract
Abort Observation	Operator	Instrument	Derived from Abstract Interrupt Observation
Stop Observation	Operator	Instrument	Derived from Abstract Interrupt Observation
Pause Observation	Operator	Instrument	Derived from Abstract Interrupt Observation
Resume Observation	Operator	Instrument	Primary

3.5 Elaborating the functional requirements for sub-systems

The high-level system scenarios identified in the previous section are next described in more detail and are referred to as the base use case descriptions (Armour & Miller, 2001). The base use case description identifies the specific behaviours and interactions that take place between actor and system within the scope of the use case's flow of events. The flow of events is a description of these activities and order in which they happen when the event that triggers the use case occurs.

The objectives in developing the base use cases are:

- Understand the primary interactions between the system and user and the system behaviours that occur as part of the use case
- Document the scope of the use case

- Start to determine non-functional requirements and assumptions associated with the use case
- Document the priority of the use case

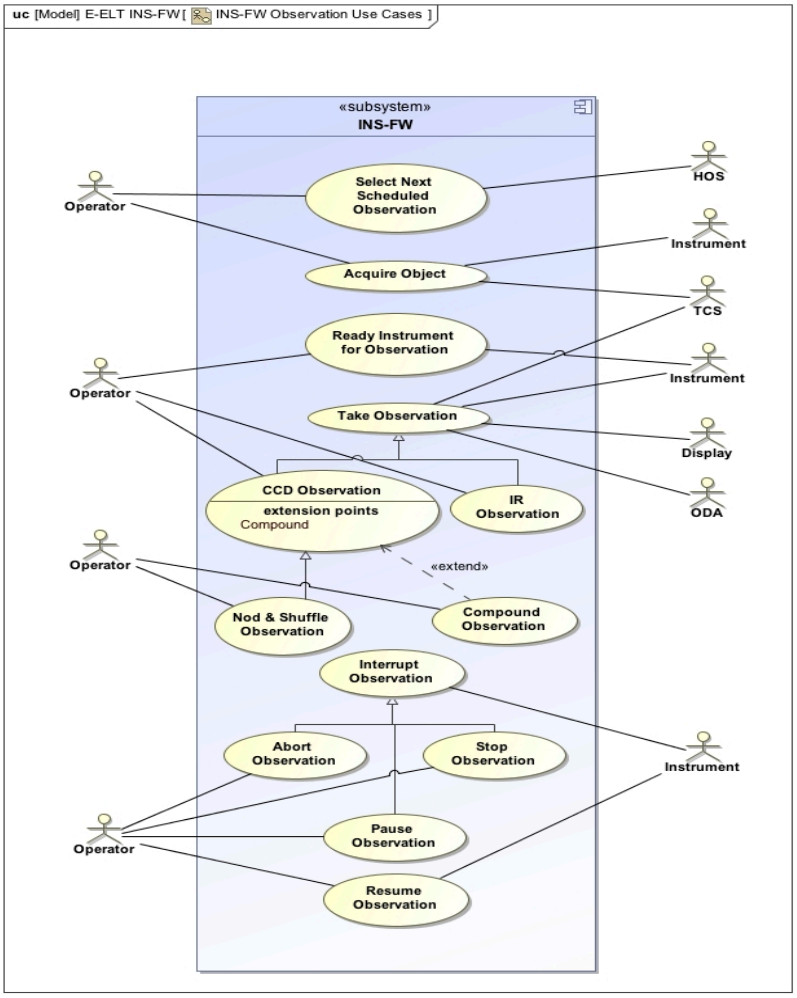

Figure 2. INS-FW Observation Use Cases

To allow these objectives to be met, the base use case provides a complete description of the *normal* (or basic) set of interactions that occur – exceptional interactions should not be covered as they cloud the understanding of the use case – these are covered later. An example use-case elaboration follows (Table 3):

Table 3. Use-case elaboration for "Ready Instrument for an Observation"

Use Case	**Ready Instrument for an Observation**
Use Case ID	UC 1.
Primary Actor(s)	Instrument Operator, Instrument
Secondary Actor(s)	Astronomer, Tester
Brief Description	The instrument is set to online state (night mode) and the setup parameters uploaded from an observation block (OB) that has been previously prepared by an astronomer or system tester.
Trigger	The Operator submits an OB to the INS-FW Sequencer and commands it to do a setup
Preconditions	• Instrument must be able to be switched to online • A valid OB must be available

Use Case	Ready Instrument for an Observation
Flow of Events	• The instrument must be ready for setup 1. Operator submits the selected OB to the INS-FW Sequencer 2. The INS-FW Sequencer interprets the OB and dispatches required setup commands to idle INS-FW instrument components concurrently if possible 3. Instrument components acknowledge reception of commands and transition to a busy state which is indicated on the operator GUI 4. The INS-FW prevents any further commands from being sent to any busy components 5. Instrument components perform the required action indicating status whilst doing so 6. Status information from each instrument component is displayed on the operator GUI 7. As components complete actions, completion status is indicated to the INS-FW Sequencer and the component transitions to the idle state. 8. The INS-FW Sequencer completes all setup commands to instrument components and indicates setup completion status to the operator
Postconditions	• Setup has either completed or failed • The instrument status is either ready or in error
Priority	High
Alternative Flows and Exceptions	TBD – enumerate any exceptional flow of events with a later iteration
Non-functional requirements	TBD – list non-functional requirements at a later iteration
Assumptions	• OB available
Issues	TBD – note any issues identified with use-case
Source	TBD – identify source of use-case

3.6 Enumerating the functional requirements for sub-systems

Functional requirements specify the operation or behaviour that the INS-FW, or part of it, must perform. These requirements have been derived from the set of use-cases and from the set of associated (higher-level) documents. The detailed functional requirements associated with each feature of the INS-FW have been itemised. These are the capabilities of the INS-FW that must be present in order for the user to carry out the services provided by the feature. Included is how the INS-FW should respond to anticipated error conditions or invalid inputs. It is important to ensure that these requirements are concise, complete, unambiguous, verifiable, and necessary.

As an example of this process the following list is shown below for the functional requirements associated with monitoring and control of an instrument.

Monitoring and Control

R.1 Instruments will require a database (the INS-FW Parameter Database) where data items can be stored, fetched and updated. Items in this database shall be publishable and shall be able to be subscribed to by any number of clients distributed anywhere on the network. This shall be achieved in a transparent fashion such that application software is unaware of the technical implementation.

R.2 The INS-FW Parameter Database shall represent the current state of the instrument and shall reside primarily on the instrument components for maximum speed of write access by each component. The database shall be reflected, when required, to other INS-FW components with minimal latency.

R.3 The INS-FW Parameter Database shall support all standard simple data types, variable length multi-dimension arrays of simple data types, variable length strings and compound data types (such as, for example, C structs or IDL definitions, etc.)

R.4 The INS-FW Parameter Database for each instrument shall be defined using a data dictionary in a manner that allows for the efficient discovery of available data items

R.5 The INS-FW Parameter Database shall support the atomic updating/publishing of non-scalar data types (such as, for example, C-structs or IDL definitions, etc.) that may have hierarchical definitions)

R.6 Transport of data into and out of the INS-FW Parameter Database shall be byte-order transparent to the application

R.7 The INS-FW Parameter Database shall store historical parameter values with quantity and frequency of retained values configurable

R.8 Component attributes set by a command, like the position for a motor, shall not be modified by INS-FW component until a new command to change the value is received

R.9 Goal and actual values of an attribute of an INS-FW component must both be stored as part of the INS-FW Parameter Database system (this may, of course, be a part of the component itself in an distributed object sense)

R.10 Set values shall be checked for validity with values falling within an admissible range or being part of an admissible set. Values shall also be syntactically correct

R.11 When a validity check fails for a particular parameter it shall be possible to automatically raise an alarm event using the INS-FW Alarms sub-system

R.12 The INS-FW Parameter Database shall support filtering using various metrics when subscribing to changes of items in the database

At time of writing, we have identified 178 functional and non-functional requirements (excluding common CIF requirements – see below) for the INS-FW. We have also noted 12 high-level business rules.

It has to be noted that CIF requirements are, for the most part, a subset of the INS-FW requirements (covering Messaging, Error Handling, Alarms, Logging and Configuration). The goal is to use the corresponding CIF subsystems below the INS-FW. The analysis of specific INS-FW requirements will influence CIF development in order to satisfy requirements that are not part of the set already identified for the E-ELT TCS

4. MODELLING ACTUAL INSTRUMENTS - VERIFYING AND TRACING REQUIREMENTS

For a more complete understanding of each of the instruments proposed for the E-ELT from a software framework development point of view, ESO have undertaken to model each of the instruments using the System Modelling Language (SysML)[2]. These models will ultimately form part of the overall E-ELT model being developed at ESO and well advanced for the TCS (see Karban et al, 2008).

This modelling has included developing structure diagrams for the instruments, internal block diagrams detailing the information flows, activity diagrams that model component behaviour, parametric diagrams that detail operating constraints and allocation of requirements to model objects for requirements traceability. An important goal of this process has been to extract common elements out of the models for each of the instruments and then to implement these common parts by constructing a catalogue containing reusable elements. The top-level structure diagram for the E-ELT instrument SysML model is shown in Figure 3 (actual instrument names removed due to current selection process).

It has to be emphasised that the process of requirements elicitation, use-case analysis and system modelling is an iterative one and refinements will need developing over a large part of the development life cycle.

5. NEXT STEPS – DEVELOPING AN ARCHITECTURE AND PROTOTYPING THE DESIGN

The next step in developing the infrastructure software for the E-ELT is deriving the overall architectural design – i.e., (quoting from Bass et al, 2003) "*something that distils away details of implementation, algorithm, and data representation and concentrates on the behaviour, and interaction of black-box elements*" This is considered a first step in designing a system that has a number of desired "properties" (i.e., requirements). Though, as pointed out in this text, from one set of requirements a bunch of different architectures can emerge given different environments - we have to be careful of current influences that may dictate a particular architectural pattern, which may not be optimal. Bass et al suggest that following principles should be followed when proceeding with this step

1. Choosing an architectural pattern is one of the architects first design choices
2. Is there a reference model for our domain?
3. And is there a reference architecture, (i.e., a reference model mapped onto software elements) for our domain that we would like to use?

This is where our experience of current software frameworks will be of advantage.

Prototyping is also an important part of the process of developing a software architecture and ultimately a design for the INS-FW. To that end ESO have been looking at promising technologies and developing prototype solutions. This has been well described in their E-ELT Technology demonstrator work (TDem) – see Karban et al 2008b. ESO have also

[2] http://www.sysml.org/

sub-contracted to external groups for initial designs of the CIF hoping ideas will emerge that can be carried through for subsequent work.

In conclusion, we expect that by following the processes described here and drawing on our previous experience, the next generation of software frameworks for the E-ELT will prove to be cost-effective, flexible and successful.

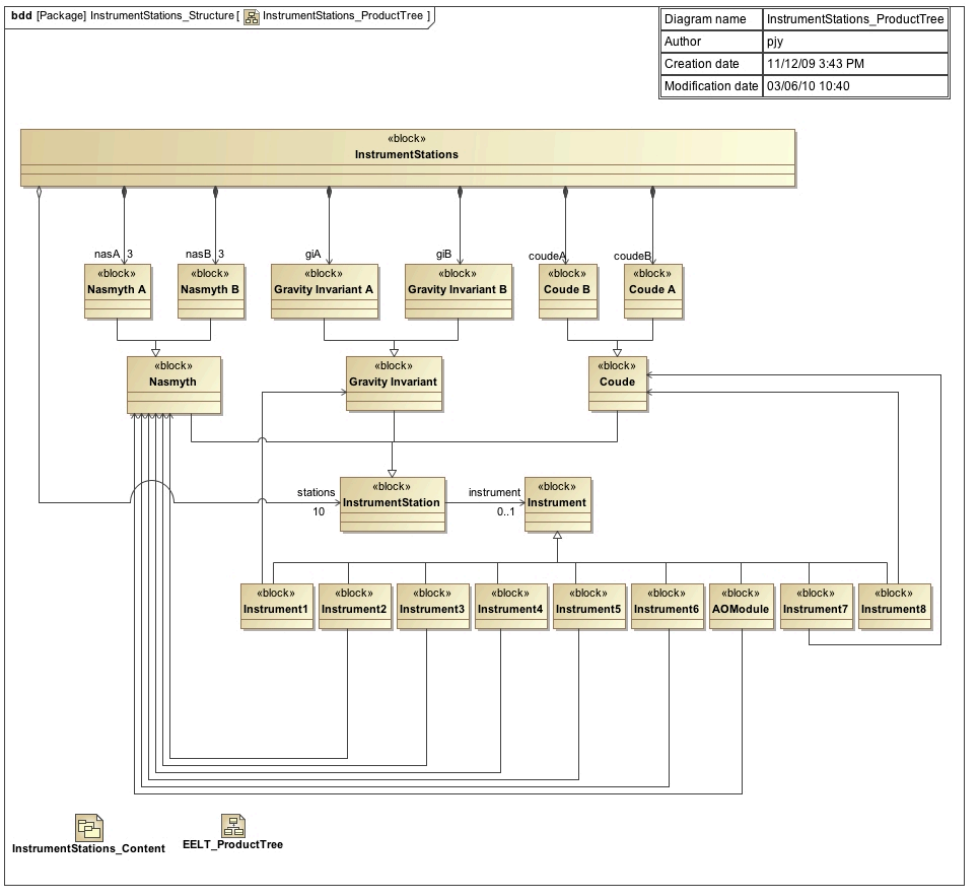

Figure 3. Example SysML model element - E-ELT Instrument Stations block diagram

REFERENCES

[1] Armour, F. & Miller, G., [Advanced Use Case Modelling: Software Systems], Addison-Wesley, Boston, (2001).

[2] Bass, L., Clements, P. & Kazman, R., [Software Architecture in Practice, 2nd ed], Addison-Wesley, Boston, (2003).

[3] Gillies,K., Nunez, A. & Dunn, J., [A New Approach for Instrument Software at Gemini], Advanced Software and Control for Astronomy II. Edited by Bridger, A. & Radziwill, N., Proceedings of the SPIE, Vol 7019, 70190Z, (2008).

[4] Karban, R., Zamparelli, M., Bauvir, B., Koehler, B., Noethe, L. and Balestra, A., [Exploring model based engineering for large telescopes: getting started with descriptive models], Modelling, Systems Engineering, and Project Management for Astronomy III, edited by George Z. Angeli, Martin J. Cullum, Proc. of SPIE Vol. 7017, 70171I, 2008 for Astronomy III, edited by George Z. Angeli, Martin J. Cullum, Proc. of SPIE Vol. 7017, 70171I, (2008).

[5] Karban, R., Chiozzi, G. Duhoux, P., Kiekebusch, M., Zagar, K., [E-ELT Technology Demonstrator Report: Software Integration], ESO internal document, E-TRE-ESO-449-0292 Issue 1, (2008).

[6] Wiegers, K.E., [Software Requirements], Microsoft Press, Redmond, (2003).