SCOS II - An Object Oriented Software Development Approach

*Martin Symonds, Steen Lynenskjold, Christian Müller.**

Computer Resources International A/S
Bregnerødvej 144
DK - 3460 Birkerød, Denmark

ABSTRACT

The Spacecraft Control and Operations System II (SCOS II), is intended to provide the generic mission control system infrastructure for future ESA missions. It represents a bold step forward in order to take advantage of state-of-the-art technology and current practices in the area of software engineering. Key features include:

- Use of Object Oriented Analysis and Design techniques
- Use of UNIX , C++ and a distributed architecture as the enabling implementation technology
- Goal of re-use for development, maintenance and mission specific software implementation
- Introduction of the concept of a spacecraft control model.

This paper touches upon some of the traditional beliefs surrounding Object Oriented development and describes their relevance to SCOS II. It gives rationale for why particular approaches were adopted and others not, and describes the impact of these decisions.

The development approach followed is discussed, highlighting the evolutionary nature of the overall process and the iterative nature of the various tasks carried out.

The emphasis of this paper is on the ***process*** of the development with the following being covered:

- The three phases of the SCOS II project - prototyping & analysis, design & implementation and configuration / delivery of mission specific systems
- The close co-operation and continual interaction with the users during the development
- The management approach - the split between client staff, industry and some of the required project management activities
- The lifecycle adopted being an enhancement of the ESA PSS-05 standard with SCOS II specific activities and approaches defined
- An examination of some of the difficulties encountered and the solutions adopted.

Finally, the lessons learned from the SCOS II experience are highlighted, identifying those issues to be used as feedback into future developments of this nature.

This paper does not intend to describe the finished product and its operation, but focusing on the journey to arrive there, concentrating therefore on the processes and not the products of the SCOS II software development.

INTRODUCTION

SCOS II

SCOS II (Spacecraft Control and Operations System II), ref. [10][11][12][13] is the latest of ESA's (European Space Agency), efforts to increase standardisation and reuse within its control systems. SCOS II has as a predecessor SCOS I which provides standard functionality for the telemetry processing chain and various data management features. These standard features such as telemetry displays, out of limits checking, database maintenance etc. were provided as a collection of middleware routines and tasks around which a mission would build its Telecommanding chain and any other mission specific components. SCOS I uses as front-end, non standard, custom built workstations connected to centralised VAX computers. An enhancement to SCOS I which has recently been made available provides the same underlying functionality but using Sun workstations connected to the VAX's.

SCOS II goes some steps further. In addition to the functions provided by SCOS I, it not only provides standard telecommanding facilities but is also designed to allow much more mission specific customisation of the kernel system. This customisation is readily available as a result of the Object Oriented approach and underlying technology adopted, and is outlined in the sections which follow.

---

* Martin Symonds (martin@msymonds.demon.co.uk), Steen Lynenskjold (steen@acm.org) and Christian Müller (cmueller@esoc.bitnet) are currently assigned to the European Space Operations Centre in Darmstadt, Germany. They have worked for **CRI** on the management, analysis, design, implementation and testing of the Application part of the SCOS II project under a contract with the European Space Agency.

The approach taken by the SCOS II project was designed to provide the maximum benefit from use of current "State of the art" tools and techniques in the field of Software Engineering. These were not chosen for their own sake, but in order to deliver very real benefits to the development lifecycle and the final SCOS II products. In particular, the use of Object Oriented Analysis and Design techniques, and a move towards an open distributed architecture based on the use of C++ running under Solaris on Sun workstations, complemented each other well. In addition, tools such as those used for user interface design and implementation helped the prototyping and user requirements definition considerably.

Probably the most important design driver was that SCOS II should be **generic**. That is, not only should it make use of the available technology and the re-usability provided by object orientation, but it should also ensure that the re-use is embedded in the design and not just the implementation.

For example, one can imagine the system needing to know about gyros, heaters and thrusters. To use the object oriented approach one could implement these as separate classes and then specialise from them in order to make different kinds of gyro, heater and thruster. The SCOS II approach however has found a way to ensure that gyros, heaters and thrusters can all be specialised from a single parent, called the System Element. The adopted client-server concept plus the distributed architecture brings a flexible system with high performance. It is these extra steps which will deliver some of the real power and benefit of SCOS II.

OOA\OOD

As well as the standard functional goals and requirements of a satellite control system, SCOS II has a number of other goals for ESA/ESOC. In particular these are centred around the concept of reuse of the software and tools used during the requirements definition, development and maintenance phases of SCOS II. SCOS II is also required to allow easy mission specific customisation of the kernel whilst providing for the mission specific components to be optionally later included into SCOS II. This is achieved by implementing a building block approach for both the design and use of SCOS II.

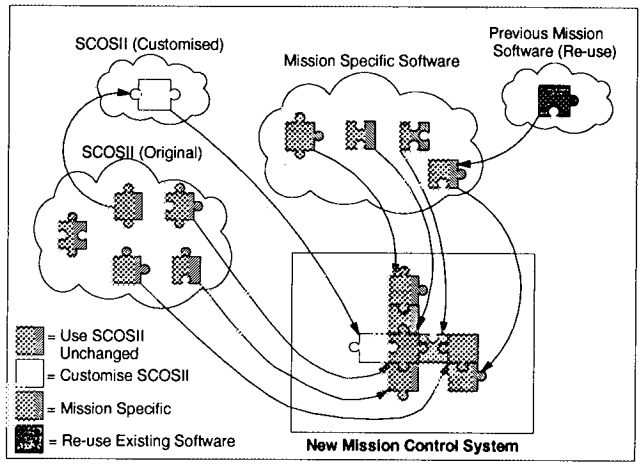

Figure 1 : SCOS II Building Block Approach

The concept behind this will allow a control system to be put together from the SCOS II supplied components, modified SCOS II components and mission specific components. This approach is illustrated in Figure 1 where the final components of the system are shown as being built from each of the various sources. This building block approach is supported by the use of C++ and the class libraries that the SCOS II project provides, allowing a "mix and match" approach to system construction as shown in Figure 1.

In addition to the goals and expected benefits for the developers, there are also changes occurring for the users. These changes include increased involvement in the analysis and design process, the capability to represent and control their spacecraft through the use of a model and changes in the physical appearance of the system.

One of the most significant changes that SCOS II users have had to come to grips with is the change in emphasis between focusing on the mechanisms used for controlling the spacecraft to focusing on the spacecraft itself. For example, the tendency in the past has been to think of commanding and monitoring of the spacecraft in terms of telecommands and telemetry, whereas the SCOS II approach encourages focus on the actual spacecraft and its components, i.e. those objects being commanded or monitored (gyros, heaters, thrusters etc.). This manifests itself primarily as a consequence of the Object Oriented Analysis and Design approach which allows the spacecraft model to be developed as part of the tasks carried out by the users when configuring the system. These modelling features allow the easy expression of physical, thermal and electrical relationships as well as abstract relationships as and when required by the users.

OBJECT ORIENTATION

The concepts of object orientation have been in the software industry for some years now but it is only recently that the tools, methods and experience have become readily available to allow the widespread take-up of this approach and the techniques it supports. The benefits of object orientation permeate the entire software development lifecycle, from the analysis of user requirements through to maintenance and operations. The major advantages of object orientation within the software development lifecycle can be summarised as follows:

- **Analysis of Problem Domain** - Allowing a better understanding of the problem domain; encouraging user/analyst interaction; providing a basis for evolution towards the design and implementation
- **Design of solution** - Encouraging identification and utilisation of underlying commonality within the problem domain; providing a means of concealing changes to the design specification; extending results of analysis phase
- **Maintenance and operations** - Promoting reuse of developed components; concealing low level code changes.

These advantages can be considered a result of the tools, methodology and languages used. In particular the object oriented concepts of encapsulation, inheritance and polymorphism allow a number of the advantages listed above to be realised.

The object oriented approach also supports an iterative lifecycle where iteration is considered part of the analysis and design process as further detail is added to the analysis/design model. Figure 2 below shows the iterative nature of the lifecycle approach taken, which can be compared with the traditional waterfall lifecycle in Figure 3.

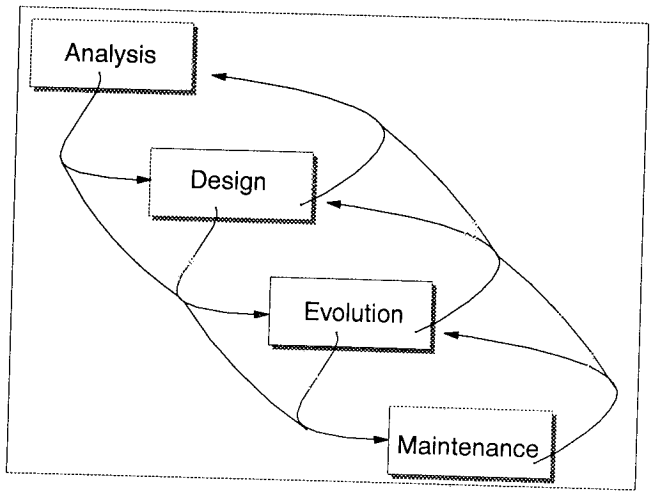

Figure 2 : Object Oriented Lifecycle

The major difference is that iteration and feedback is a fundamental part of the object oriented lifecycle, whereas for the traditional waterfall lifecycle, this feedback is generally only permitted to rectify errors. The object oriented approach allows the analysis results to be gradually expanded and refined with successive layers of detail until the design is complete. It is hence of outmost importance to

define each iteration and its products as part of the planning cycle.

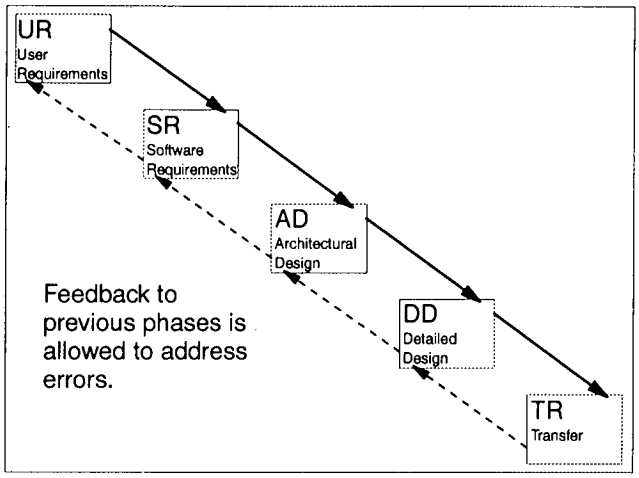

Figure 3 : Traditional Waterfall Lifecycle

SCOS II

In order to satisfy the demands placed upon SCOS II, the project was approached in two phases. In the first phase, the technology to be used was proven in terms of functionality and performance, and the initial analysis work was carried out in conjunction with a significant amount of user interface prototyping.

Once the technology had been proven and the initial analysis performed, the project moved into its main development phase which saw the underlying technical services being provided and the analysis/prototyping activities moving forward into design and implementation of those generic parts of the system identified in the analysis.

Whilst the initial phase was not a pilot project as such, it did allow the project team to get to grips with the technology, tools and problem domain, providing them with the means to determine the route to the system goals as part of the second phase.

The project team structure saw a peak of some 20 software engineers. Of these, 5 were client staff responsible for the overall management, technical management and system testing support. Industry was represented by two consortia, each of 3 companies with clearly defined responsibilities. The Application Team was responsible for providing the analysis of the problem domain and for ensuring that the users functional requirements were satisfied. This team also carried out extensive functional prototyping and is responsible for delivering SCOS II applications. The second industry team was responsible for providing the low level technical infrastructure such as software to handle the transmission and caching of data across the network.

DEVELOPMENT APPROACH

In describing the development approach, it is necessary to understand the standard ESOC activities, how these activities were mapped on to the phases adopted by SCOS II, the modified lifecycle used by SCOS II and what were the key features of the development under these constraints.

Activities

ESA software development projects are developed according to the ESA Software Engineering Standards, ref. [9]. These standards recognise five phases of the development lifecycle known as:

- **User Requirements Definition** - definition of the problem domain to be solved by the system to be procured
- **Software Requirements Definition** - Analysis of user requirements to define a model to allow satisfaction of these requirements
- **Architectural Design** - Design of the hardware and software architecture including data and control flow
- **Detailed Design** - Design, code and test of the system design

- **Transfer** - Installation of software in target environment; performance of acceptance testing

These have traditionally been performed using the traditional waterfall lifecycle shown in Figure 3.

Whilst this is a well proven method, it has a number of difficulties and inconsistencies. These are emphasised when attempting to use this approach in an object oriented environment. The major difficulty is that in the waterfall lifecycle, the output from one phase is the major driver for the following phase, and to a large extent stands alone. The object oriented approach however encourages successive refinement of the initial analysis model right through to the code, without being able to easily produce the corresponding breakpoints of a traditional lifecycle. This is demonstrated by Figure 4 which shows how the relationship between the successive phases of a traditional approach is less closely coupled to its previous phase than that of an object oriented lifecycle.

With the waterfall approach, there are clearly defined deliverables at the end of each phase, which stand alone. With the object oriented approach, each iteration sees further refinement and not necessarily a specific stand alone product. Each iteration product should be defined in a manner that it is tangible; hereby giving the management the necessary information to monitor progress.

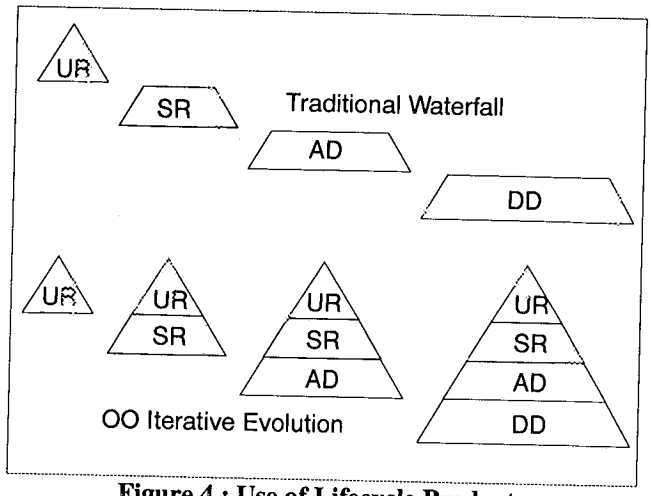

Figure 4 : Use of Lifecycle Products

Phases

The SCOS II approach required that the development be object oriented yet maintain, wherever possible, a correspondence to the ESA PSS-05 phases and deliverables. This was not easy and became more challenging as the project progressed.

The prototyping and analysis phase corresponded closely in some ways to the traditional lifecycle with the SCOS II development team producing an object oriented SRD (Software Requirements Document), ref. [2]. It was found that the nature of the object oriented analysis was such that the SRD activities could in fact be performed in parallel with the URD, with final SRD updates lagging behind the final release of the URD. During this phase, extensive iterative prototyping took place in order to:

- help elicit user requirements
- define user interfaces.

This proved to be a valuable exercise for the users.

The methodology followed for this analysis phase was the Coad/Yourdon method, ref. [5][6][7]. The object/class diagrams were created using the OMTool product which uses the Rumbaugh notation, ref. [3].

The Design and Implementation phase saw SCOS II covering the traditional AD/DD activities. Once more the nature of the object oriented approach is such that it was found that the SRD was more detailed than a traditional SRD and addressed a level of detail not normally found until AD activities. Similarly, the AD documentation progressed to a point where traditional DD issues were being addressed. It was also noted that the coding and detailed design activities were highly iterative, allowing the design and software to evolve together and to take into account feedback from users. Integration however has been more of a continuous process rather than one which progresses in clearly defined stages.

The next phase of the project which will commence in late 1994, will be to continue roll-out of the SCOS II kernel in readiness for customisation and enhancement by its first client missions. These deliveries will consist mainly of collections of C++ class libraries that will be used by the client missions as a basis for their custom and mission specific software development.

Lifecycle Considerations

The mismatch between the traditional lifecycle and that encouraged by the more iterative object oriented lifecycle continues to be a source of frustration. It is not easy to present documents for external review that correspond to some degree with the contents of traditional deliverables of that phase. Whilst less detail could have been documented during the SR and AD phases, the nature of the approach stimulates an analysis philosophy that repeatedly drops down into detail and back up again. It would be inefficient to ignore or document this information in another fashion.

Whilst SCOS II has produced documentation for review, such as the SRD, it has always been clear that the level of detail contained in these documents has generally been higher than the traditional documents. This reflects the lifecycle comparison diagram in Figure 5. This is also discussed in ref. [4].

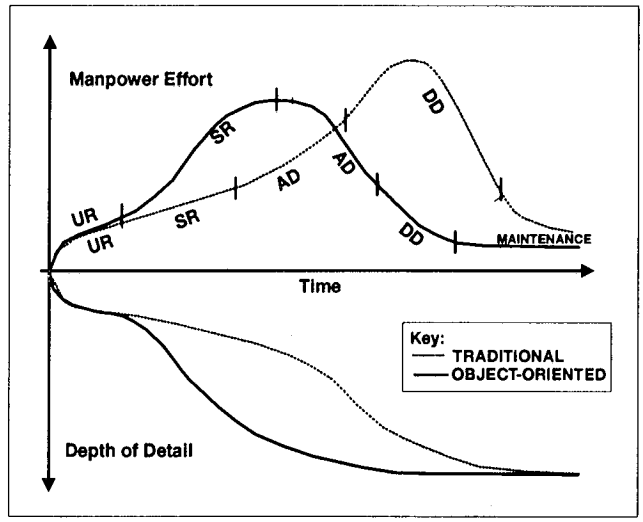

Figure 5 : Comparison of Traditional vs. Object Oriented Lifecycle Phases

Figure 5 shows some interesting comparisons between the traditional lifecycle and the object oriented lifecycle. In particular it demonstrates the OO approach reaching the same level of detail overall, but dropping down much sooner. Similarly, the corresponding amount of effort for an object oriented approach seems to occur rather earlier in the development cycle with the maintenance level is expected to be less.

SCOS II was able to take advantage of the possibility of overlapping phases. Thus whilst the UR/SR/AD/DD phases have overlapped this has not appeared to hinder development at all. This is something of a two edged sword; on the one hand it allows rapid progress towards an initial version/prototype, while on the other it does make the project management more complex.

Key Features

To summarise, the key features of SCOS II which have made it a success include:

- ***Prototyping*** - This helped considerably to elicit requirements, define interfaces and to demonstrate progress to the users.
- ***The iterative approach*** - Allowing frequent tangible results during both the analysis and design phases.
  - ◊ High level (Analysis and Design) - Manifested through successive refinement of the analysis model and refined user requirements.
  - ◊ Low level (Coding and Delivery) - Allowing successive deliveries to provide increased functionality.
- ***Object Orientation***
  - ◊ User interaction / co-operation (Through the Analysis Model) - Providing increased visibility of the design process, for the users and increased visibility of the problem domain for the developers.
  - ◊ Software Modularity - the implementation of the building block concept providing clean mechanisms for mission specific control systems.
- ***Management approach***
  - ◊ 3 groups (client and two teams from industry) - Allowing diverse skills to be brought to bear on a challenging, state of the art project.
  - ◊ split into technical and applications areas - Allowing clearly defined responsibilities
  - ◊ one infrastructure (bottom up) - Starting from the available technology and providing services for the applications.
  - ◊ one requirements (top down) - Starting from the requirements and implementing using the provided infrastructure services.

CONCLUSION

SCOS II is now well on the way to completion. It is a suitable opportunity to take a look back over the past couple of years and with the benefit of hindsight, draw some conclusions from the route that we have travelled.

The project may cost some 50% less than its predecessor infrastructure (SCOS I and MSSS)It is clear that the approach, technology and tools used have led to greater productivity in many ways.

The extent to which the benefits of ease of maintenance and later re-use will be realised, remains to be seen in client project applications. Based on the experience of flexibility to change and extent of re-use throughout the development phase, we have considerable confidence that this will be achieved

In retrospect it would have been immensely useful to have been able to develop a small pilot project. This would have enabled a number of management, analysis, design, implementation and standards issues to be resolved before SCOS II commenced. As it was these had to be addressed as part of the ongoing project work and sometimes distracted and indeed disrupted progress. To tackle a project of this nature and complexity where little appropriate expertise was available, and to add an increased level of complexity by making the SCOS II goal a generic system, is a high risk strategy. That this strategy is starting to pay off is a remarkable tribute to the skills and dedication of the people involved in the project.

REFERENCES

[1] SCOS II User Requirements Document, ESOC DOPS-SYS-URD-001-AMD, Issue 3, February 1994.

[2] SCOS II Software Requirements Document, ESOC SCOS II-SYS-SRD, Issue 0.6, June 1994.

[3] Object Oriented Modelling and Design, Rumbaugh et. Al, Prentice Hall 1991.

[4] Object Oriented Design with Applications, Grady Booch, Benjamin Cummings 1991.

[5] Object Oriented Analysis, Peter Coad/Edward Yourdon, Prentice Hall 1990.

[6] Object Oriented Design, Peter Coad/Edward Yourdon, Prentice Hall 1991.

[7] Object Oriented Programming, Peter Coad/Jill Nicola, Prentice Hall 1993.

[8] Modelling the World in States, Sally Schlaer/Stephen J. Mellor, Prentice Hall 1992.

[9] ESA Software Engineering Standards - Issue 2, ESA PSS-05-0 Issue 2, ESA Publications Division, February 1991

[10]SCOS II: ESA's New Generation of Mission Control Systems - The User's Perspective, P Kaufeler, M Pecchioli, I Shurmer, ESOC - these proceedings.

[11]A New Communication Protocol Family for a Distributed Spacecraft Control System, A Baldi, M Pace, ESOC - these proceedings.

[12]SCOS II: ESA's New Generation of Control Systems, M Jones, N Head, K Keyte, P Howard, S Lynenskjold - these proceedings.

[13]SCOS II OL: A Dedicated Language for Mission Operations, A Baldi, Dennis Elgaard, S Lynenskjold, M Pecchioli - these proceedings.

Systems Development

4. Modeling

* *Presented in Poster Session*