

# OBOSS - II

## Final Presentation

Presented by

Dr. Ulrich Denskat

Stefan Flemmig

Dornier Satellitensysteme GmbH

D-88039 Friedrichshafen

Germany

Fax: :(+49)7545/8-3332

Email: [Ulrich.Denskat@dss.dornier.dasa.de](mailto:Ulrich.Denskat@dss.dornier.dasa.de)

Email: [Stefan.Flemmig@dss.dornier.dasa.de](mailto:Stefan.Flemmig@dss.dornier.dasa.de)

## Scope of Work related to Instrument Control Units

- Study the applicability of the OBOSS data handling service concept in the ICU domain with the ICU architecture being based on the TSC 21020E DSP.
- **Way to go:**
  - Define ICU reference architecture based on OBOSS Data Handling Architecture.
  - Port subset of OBOSS PUS services to DSP based ICU for proof of feasibility.
  - Produce operational demonstration scenario – preferable a combined demonstration including the Data Handling System and the prototype Instrument Control Unit.

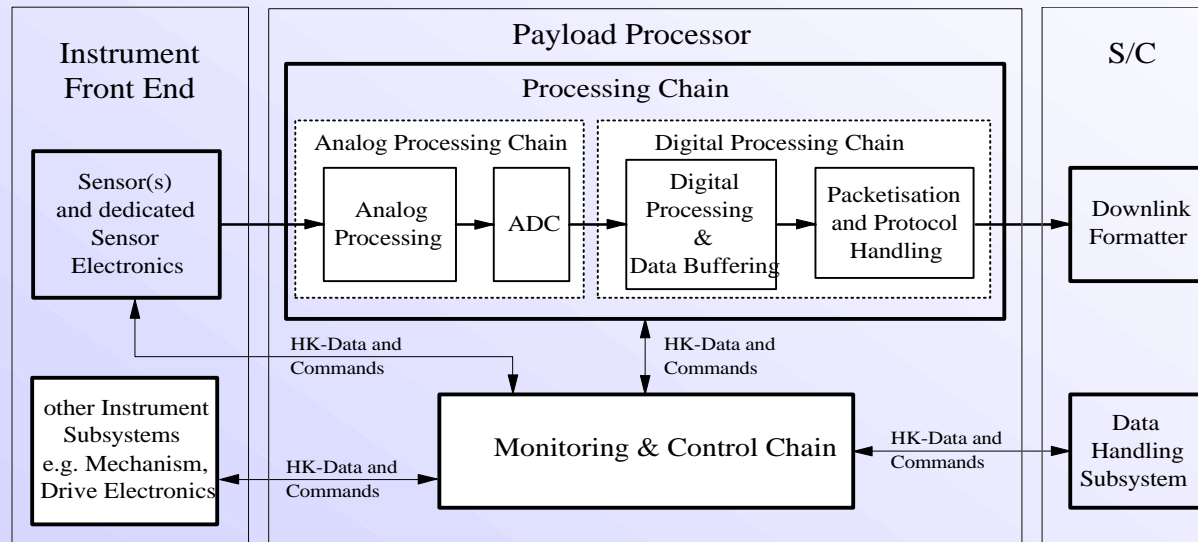
## Function and Purpose

The ported software will serve as the monitoring and control software for instruments.

It supports instrument control and data exchange with the platform.

The OBOSS ICU software will be embedded into a hierarchical command & control structure whereby the ICU forwards commands from the data handling system to the instrument and performs the monitoring of instrument parameters.

## Standard ICU Architecture



L\_ARCH2.DRW , 31.03.98

The OBOSS ICU software provides for the monitoring and control chain of the payload processor.

The figure also shows the interfaces of the ICU software within the spacecraft/payload system (i.e. the data handling system, instrument subsystems and the processing chain).

The payload processing chain could physically be placed on the same processor as the monitoring and control software if resource requirements allow.

## Ported Services

The ICU software is providing functionality following the PUS services for:

- On-Board Traffic Management
- Event Reporting
- TC Verification
- Housekeeping and Diagnostics
- Function Management
- Device Level Commanding
- On-Board Monitoring

## Lessons Learned

### In respect of working with Unix DSP simulator

First idea of using a Unix DSP simulator was dropped down because of problems with both DSP 21020 simulator (e.g. impossible to write a port file) and DSP 21060 simulator (bugs during operation)

### In respect of working with Virtuoso on the Unix side

Eonics claims to support all kinds of platforms

For Unix we would have to use a specific configured board support package

It experienced that it was much easier to work on a PC

## Was transfer Ada to C the right way to go ?

In terms of architecture

Plus :

- Copying the global architecture from the Ada code optimised time to develop an own architecture.
- Functionality of software was easy to understand while porting the code.

General :

- It was much easier to have a given architecture to transfer than to develop a new one.

## In terms of Coding

### Plus :

- Ada body-files could easily be transformed in C-source-files.
- Ada specification-files could easily transformed in Header-files.
- Function- and procedure-names could be kept (if not overloaded).
- Most of the parameter names (data types, records) could be kept.

### Minus :

- Coding as near as possible to Ada code without looking for needs of the new platform resulted later in memory problems, because :
  - ➔ a lot of functions call only other functions. Parameters of function-calls are put on the stack.
  - ➔ Some really big records (kilobyte-sizes) are used which are put in the parameter list of function-calls.
- To cope with this problem the code was cleaned up and arrays are reduced to fit within stack sizes.
- Errors may result from overflow of arrays.

## In terms of Coding

### Minus :

- New functions had to be written for all Ada library routines (e.g. unchecked conversion).
- Generic implementations in the Ada code had to be replaced (e.g. map\_types).
- Instantiations had to be replaced (e.g. app\_queue).
- Pre-defined Ada data types had to be replaced by C data types.
- Some bit-shift operations had to be included to handle equivalent operations in the Ada code which operate on small portions of words and bytes.
  
- C offers a simpler language than Ada. In the ICU implementation is no need to reproduce all data-types which are present in the Ada code.

### General :

- This software is a prototype which doesn't make full use of C-capabilities.
- Memory problems on DSP-memory could have been avoided at an earlier stage.

## Some Metrics

### Lines of code

General : 13,200

Added or changed code (with respect to Terma DHS) in terms of architecture, without looking at changes Ada → C :

Interface : 160

Shift Operations : 350 had to be included to handle working with single bits (e.g. needed on packet header)

Replacing Ada generics by copies 150

Tasks : 210

Virtuoso : 190 without include files and kernel

Total : 1,060

Changes to architecture about 8%

## Memory Usage of ICU software

Size of exe-file : 382 kByte

Program Memory 27403 Words for all instructions

Data Memory 26914 Words for all global variables

Local variables are put on the heaps of the tasks

## Comparison with other ICU architectures

	<b>OBOSS ICU SW</b>	Envisat Sciamachy	Envisat PEB SW	Metop Ascet	Rosetta*
Lines of Code	<b>~13_200</b>	~28_000	~35_000	~28_000	
Instructions/ Words	<b>27_403</b>	64_483	77_337	79_712	153 kWords
Constants/ Words		28_978	59_649	8_565	
Data/ Words	<b>26_914</b>	96_365	188_847	74_999	392 kWords
Heap/Words	<b>66_000**</b>	(inc. Heap)	32_786		
Supported PUS similar services	<b>7</b>	12	13	11	15 + 6 other

\*Software for Rosetta doesn't exist by now. Memory budget is estimated.

\*\* This is not optimised. Optimisation may reduce it to under 50 % (estimated).

## DSS view for implementation of ICU (compared with Metop Ascet and MPU)

	LOC Ascet	LOC OBOSS ICU	Estimated Functional Reuse Potential
• Monitoring	1460	3200	90 %
• Onboard Traffic Management	1170	700	100 %
• Function Management	390	320	90 %
• Onboard Scheduling (not included in ICU-SW)	160		100 %
• Telecommand Verification		310	100 %
• Event Reporting		200	100 %

## DSS view for implementation of ICU

	LOC Ascat	LOC OBOSS ICU	Estimated Functional Reuse Potential
• Platform Specific	7930	3100	60%
• Application Specific	5430	1300	0%
(e.g. Measurement Table Management, Measurement Sequencing, Control Functionality)			
• Anomaly Handling	540		
(not contained in PUS) may be realised with PUS Function Management			
• History Reporting	1600		
(not contained in PUS) may be realised with PUS Onboard Storage and Retrieval			

## DSS view for implementation of ICU

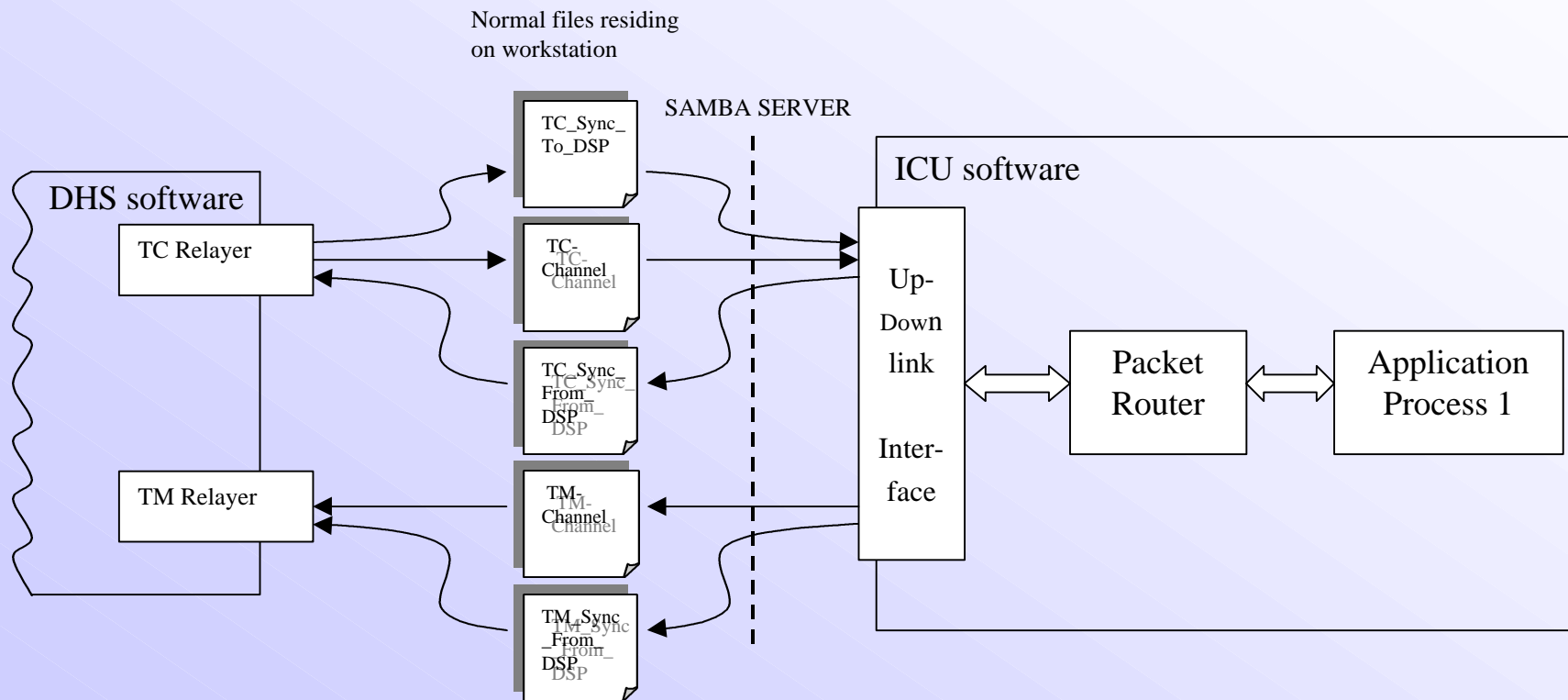
	LOC Ascat	LOC OBOSS ICU	Reuse Potential
inspected lines of code	19040	9130	
reusable lines of code		6238	68 %

## Future Directions

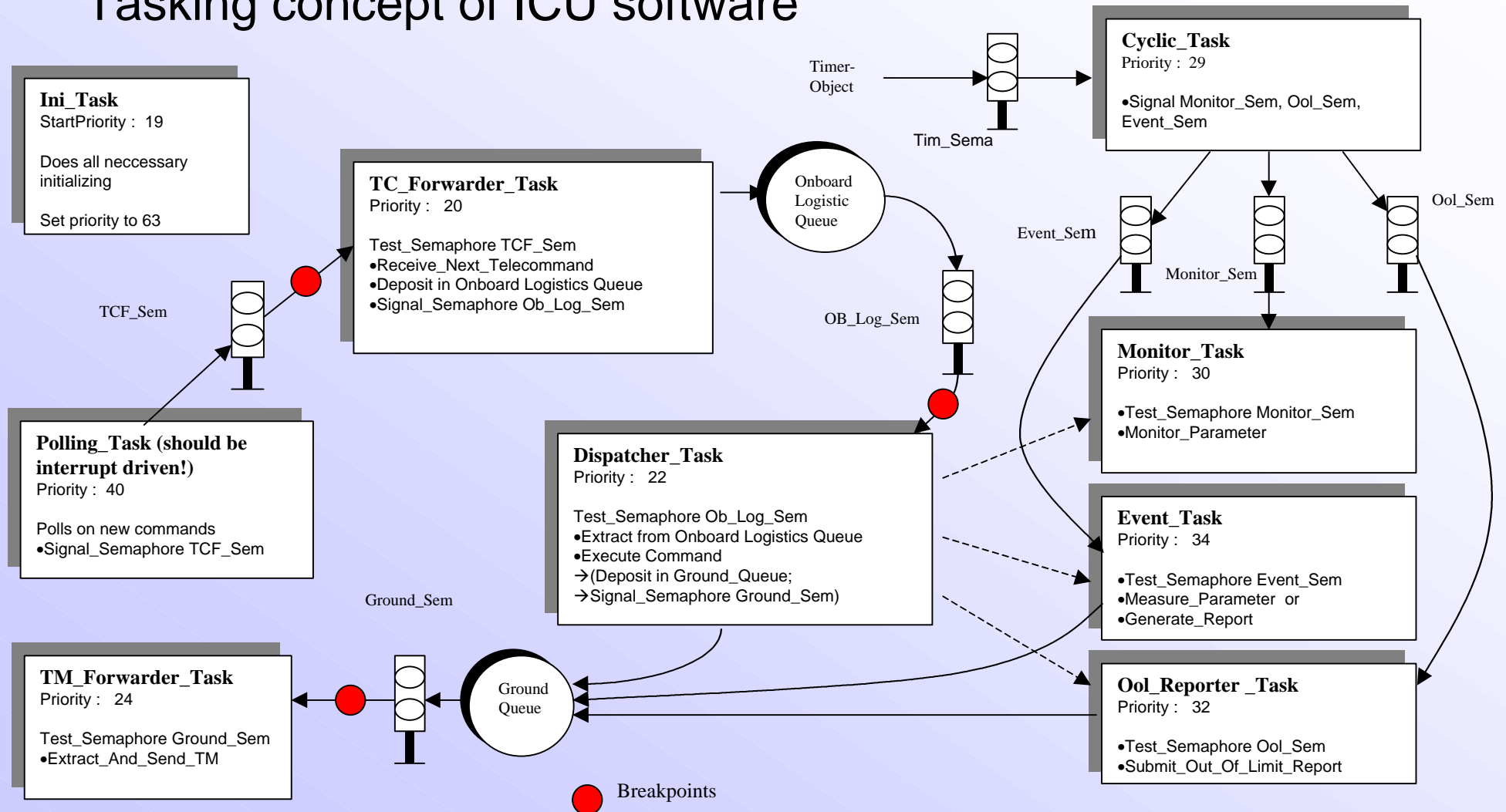
Goal : Generic ICU services ready for operational use

- PUS service extension listed in the Domain Analysis (cf. OBOSS-II Web repository)
- Convert the prototype into an optimised software by use of advantageous C features
- Optimisation of code in direction of heap sizes and execution speed
- Porting software to a physical DSP platform for real performance assessment
- Strict verification approach

## Set-up of ICU part



## Tasking concept of ICU software



## Possible Responses from ICU

	<b>Exception Report</b>	Verification Packet	Housekeeping Report Definition Report	<b>Housekeeping Parameter Report</b>	Monitoring List Report	<b>Out of limit Report</b>
Description	Generated if an error occurs in the TC, which make execution of TC impossible	Generated on acknowledged states of execution	Reports the parameters in the housekeeping list	Reports the measured values of parameters for housekeeping	Reports the parameters and their limit settings in the monitoring list	Reports parameters which are out of limit
Generation mode	Sporadic on event	Generated for each TC on request	On request	Cyclic	On request	Cyclic on event