**European Space Agency Contract Report**
**ESTEC Contract No.: 13309/98/NL/MV**

# Procurement of a SDL Model for Behavioural Validation of MSL

## Final Report

**SMSL-RP-001-BSSE**

**Issue: 1**

**Revision: 0**

Dr. Rainer Gerlich BSSE System and Software Engineering
D-88090 Immenstaad (Germany)

November 5, 1999

This report was prepared for the European Space Agency under ESTEC Contract No. 13309/98/NL/MV.
Responsibility for the contents resides in the author or organisation that prepared it.

**Dr. Rainer Gerlich**

BSSE System and Software Engineering

# Document Approval Sheet

| | |
|---|---|
| **Project:** | SMSL |
| **Document Title:** | Procurement of a SDL Model for Behavioural Validation of MSL Final Report |
| **Document Number:** | SMSL-RP-001-BSSE |
| **Issue:** | 1 |
| **Revision:** | 0 |
| **DRD** | --- |

| | | | |
|---|---|---|---|
| Prepared: | ..... (Dr. R. Gerlich, BSSE) | Date: | ..... |
| Approved: | ..... (Dr. R. Gerlich, BSSE) | Date: | ..... |
| Released: | ..... (Dr. R. Gerlich, BSSE) | Date: | ..... |
| Released: | ..... (Jean-Loup Terraillon, ESTEC) | Date: | ..... |

Dr. Rainer Gerlich BSSE System and Software Engineering

Auf dem Ruhbuehl 181
D-88090 Immenstaad
Germany

Phone: +49/7545/91.12.58
Mobile: +49/171/80.20.659
Fax: +49/7545/91.12.40
email: gerlich@t-online.de
homepage: http://home.t-online.de/home/gerlich/

ESTEC Project Officer: Mr. Jean-Loup Terraillon, TOS-EME

# Document Change Record

| Iss./Rev. | Date | DCN.-No. | Pages Affected |
|---|---|---|---|
| Draft / 0 | 1999-08-15 | | Established |
| 1 / 0 | 1999-11-05 | | Editorial changes + section 8.4 added |

## LIST OF CONTENTS

**1 INTRODUCTION 1**

**1.1 Purpose of the document 1**

**1.2 Scope 1**

**1.3 Definitions, Acronyms and Abbreviations 2**

**1.4 References 3**

1.4.1 Applicable Documents 3

1.4.2 Reference Documents 3

**1.5 Overview of the Document 4**

**2 MSL SYSTEM OVERVIEW 5**

**2.1 General Overview 5**

**2.2 Configuration and Functions 5**

**2.3 Software Model 8**

**2.4 Control and Data Flow 14**

**2.5 Standardization of Intertask Communication 18**

**2.6 Principal Components of the Software 19**

2.6.1 Command Handler 19
2.6.1.1 Command formats 20
2.6.1.2 Dispatching Handling 20
2.6.1.3 Command control 21

2.6.2 Data Acquisition and Data Handling Processes 21
2.6.2.1 Data Handling Function 21
2.6.2.2 Data Supervisor Function 22

2.6.3 Telemetry Dispatcher 22

2.6.4 Error and Message Logging Function 23

2.6.5 Interface Handling Processes 23
2.6.5.1 ETHERNET Daemon Process 23
2.6.5.2 MIL Bus Interface Daemon Process 23
2.6.5.3 Digital I/O Daemon Process 23
2.6.5.4 Serial I/O Daemon Processes 23
2.6.5.5 Analog Input Daemon Processes 23
2.2.6.6 Mass Memory Daemon Process 24

**3. INTRODUCTION TO ISG 25**

**3.1 The Idea and Its Realisation 25**
3.1.1 Overview 25
3.1.2 The Organisation 26
3.1.3 Feedback and Visualisation 26
3.1.4 Real-Time Processing 27
3.1.5 Fault Injection 27
3.1.6 Integration with Existing Software 27
**3.2 The Principal Elements of the Organisation 27**
3.2.1 Processes and Devices 27
3.2.2 States 27
3.2.3 Data Processing and Commanding 28
3.2.4 Channels 29
3.2.5 Resources 30
3.2.6 Data Exchange Format 31
3.2.7 Interfaces 32
**3.3 How to Establish a New Project 32**
**3.4 Formal Specification of Behaviour and Performance 33**
**3.5 Semantics 36**
3.5.1 Incoming Commands 36
3.5.2 Outgoing Commands 36
3.5.3 Mapping of Logical Channels onto Physical Channels 38
3.5.4 Checks 39
3.5.5 Extensions of the Concept 39
**4. THE VERIFICATION AND VALIDATION STRATEGY 41**
**4.1 Rational 41**
**4.2 The Cornerstones 42**
**4.3 The Practice 42**
**4.4 More Support by Formal V&V 44**
**4.5 Portability 44**
**5. THE INPUTS REQUIRED FOR THE V&V ACTIVITIES 45**

**6. THE PERFORMED V&V ACTIVITIES 50**

**6.1 Overview 50**

**6.2 Categorisation of the Identified Errors 51**

6.2.1 Visual Checks and Automated Checks at Pre-Run-Time 51

6.2.2 Errors Detected by Execution 52

6.2.3 Errors Detected by Coverage Analysis 53

6.2.4 Errors and Anomalies Detected during Stress Testing 54

6.2.5 Observations Made In Case of Error Injection 54

**6.3 The V&V Phases 54**

**6.4 Effort 55**

6.4.1 Start-up 55

6.4.2 Follow-On 55

6.4.3 Feedback for Improved Support 55

6.4.4 Update of Coomand Procedure Table vs. Corrective Maintenance of Infrastructure 55

**7. EVALUATION OF THE RESULTS 56**

**7.1 The Graphical Presentation Capabilities 56**

**7.2 Comparison of Expected and Observed Behaviour 65**

**7.3 Consistent Correlation of External with Internal Commands 66**

**7.4 Automated Stimulation with External Commands 67**

**7.5 Check of Timeout Conditions 68**

**7.6 Coverage Analysis 69**

7.6.1 Coverage of Command Lines 69

7.6.2 Coverage of States 73

7.6.3 State Transitions 76

7.6.4 Exception Report 83

7.6.5 Error Injection Report 83

**7.7 Performance Analysis 84**

7.7.1 CPU Utilisation 84

7.7.2 Network Utilisation Report 97

7.7.3 Timer Report 99

7.7.4 CPU Load Calibration 100
7.7.5 Response Time Report 101
7.7.6 Report on MSC Generation 103
7.7.7 Command Buffer Report 103
**7.8 The "Stress Testing" Case 111**
7.8.1 The Exception Report 111
7.8.2 The Command Buffer Report 112
7.8.3 The report on CPU Utilisation 114
7.8.4 The Timer Report 126
**7.9 The "Error Injection" Case 127**
**8. CONCLUSIONS 129**
**8.1 General Conclusions 129**
**8.2 Specific Conclusions 130**
**8.3 Assessment on the V&V Environment 130**
**8.4 Final Remarks on Target Platform Activities 132**
**9. RECOMMENDATIONS AND GUIDELINES 133**

**Acknowledgement:**

The author thanks Mr. Jean-Loup Terraillon and Mr. Michael Birk for the support they gave to the project.

# 1 Introduction

## 1.1 Purpose of the document

This document describes the verification and validation concept, procedures and activities which were performed in the course of this project "Procurement of a SDL Model for Behavioural Validation of MSL" (SMSL) in order to demonstrate the feasibility of early system validation in the context of a real project for which MSL could be selected due to the interest of and support given by the MSL project.

## 1.2 Scope

The scope of this project is to support the MSL project for verification and validation (V&V) of the MSL software by applying a formal concept as suggested by the EaSyVaDe [RD3b] approach and its actual improvements and to demonstrate the benfits of this concept in the context of a real on-board project.

The goal is to establish a representative behavioural model of MSL (Material Science Laboratory) within the EaSySim II environment based on the use of Finite State Machines (FSM) and a formal specification of performance properties and topology, to apply an automated test approach including fault injection, and to perform system validation activities.

This approach aims

- to automate the construction of a behavioural skeleton of the operational software and of the test scenarios based on the specified data flow and
- to achieve a 100% coverage of input, output and states under consideration of fault injection and performance aspects.

The MSL is one of the scientific payloads installed in the US LAB of the International Space Station.

It serves for scientific research on different material under microgravity conditions. In principle MSL is a furnace that allows to melt samples at temperatures up to 1700°C and to survey and influence the crystallization and solidification.

The software developed for MSL provides the functionality to control the facility and the processing of the samples.

### 1.3 Definitions, Acronyms and Abbreviations

| | |
|---|---|
| BSW | Basic Software |
| CPT | Command Procedure Table |
| EM | Engineering Model |
| FM | Flight Model |
| FSM | Finite State Machine |
| GPBI | General Purpose Instrumentation Bus |
| HK | housekeeping |
| HW | Hardware |
| I/O | Input/Output |
| ISG | Instantaneous System and Software Generation |
| KT | Kayser-Threde GmbH, Munich |
| MSC | Message Sequence Chart |
| MSL | Material Science Laboratory |
| MSRR | Material Science Research Rack |
| O/B | on-board |
| O/G | on-ground |
| OS | Operating System |
| RSIM | Resource, Interface and Scenario Manager |
| SDL | System Description Language |
| SMSL | Procurement of a SDL Model for Behavioural Validation of MSL |
| SPLC | Standard Payload Computer |
| SRM | Science Reference Model |
| SW | Software |
| UDC | User-Defined Command |
| UDF | User-Defined Function |
| VME | VERSAM Module Europe |
| V&V | Verification and Validation |

## 1.4 References

### 1.4.1 Applicable Documents

[AD1] ESA PSS-05-0, Issue 2, Feb. 1991, ESA Software Engineering Standards

[AD2] BSSC(95)2 Issue 1, Draft 2 (Dec. 1995), "Guide to applying the ESA software engineering standards to small software projects”

[AD3] BSSC(98)1 Issue 1, (March. 1998), "Guide to applying the ESA software engineering standards projects using object-oriented methods”

[AD4] MSL - Electrical Subsystem, Software Architectural Design Document

MSL-SP-003-KT, 3 / 2

### 1.4.2 Reference Documents

[RD01] MSL - Electrical Subsystem, Command Interface Specification
MSL-SP-032-KT, Draft / -

[RD02] HRDMS (Highly Reliable DMS and Simulation), ESTEC contract no. 9882/92/NL/JG(SC), Final Report, Oct. 1994, Noordwijk, The Netherlands

[RD3a] R.Gerlich, V.Debus, Ch.Schaffer, Y.Tanurhan: EaSyVaDe: Early Validation of System Design by Behavioural Simulation, ESTEC 3rd Workshop on "Simulators for European Space Programmes" Noordwijk, November 15-17, 1994

[RD3b] OMBSIM (On-Board Mangement System Behavioural Simulation), ESTEC contract no. 10430/93/NL/FM(SC), Final Report Nov. 1995, Noordwijk, The Netherlands

[RD04] DDV (DMS Design Validation), ESTEC contract no. 9558/91/NL/JG(SC), Final Report Dec. 1996, Noordwijk, The Netherlands

[RD05] EaSySim II, 1996-1999, Rainer Gerlich BSSE System and Software Engineering, Auf dem Ruhbuehl 181, D-88090 Immenstaad, Germany

[RD06] Executable Specifications with particular application to spacecraft control and data systems, 2nd Round Table Proceedings, December 10-11, 1996, ESTEC, Noordwijk, The Netherlands

[RD07] OPAL, Protocol Validation, BSSE, 1997, unpublished

[RD08] CADIS (Central and Remote Data Acquisition and Distribution Integrated System), 1997-1999, Rainer Gerlich BSSE System and Software Engineering, Auf dem Ruhbuehl 181, D-88090 Immenstaad, Germany

[RD09] ObjectGEODE SDL-Tool, Verilog, 150 rue Vauquelin, F-31081 Toulouse Cedex, France

[RD10] SDT, TeleLogic AB, PO Box 4128, S-20312 Malmö,Sweden

[RD11] Message Sequence Chart (MSC), Z.120 (10/96), ITU General Secretariat - Sales Section, Place de Nations, CH-1211 Geneva 20

[RD12] ISG User's Manual, 1999, Rainer Gerlich BSSE System and Software Engineering, Auf dem Ruhbuehl 181, D-88090 Immenstaad, Germany

## 1.5 Overview of the Document

The document is structured as follows:

- Chapter 2 gives an introduction into the MSL application

  The contents of this chapter is an extract of [AD4].

  We thank the MSL project for the permission to include this contents directly.

- Chapter 3 introduces the ISG software approach

  This is an extract from the ISG User's Manual

- Chapter 4 describes the applied V&V approach and gives background information on the history
- Chapter 5 describes which input is required to construct the MSL software automatically and to apply the V&V concept of ISG
- Chapter 6 lists the performed V&V activities and presents a first analysis of the activities
- Chapter 7 includes the detailed figures, which have been collected during the V&V activities, and analyses and comments them
- Chapter 8 analyses the activities and the achieved results and gives conclusions
- Finally, chapter 9 gives recommendations and guidelines for future use of the concept and the environment.

# 2 MSL System Overview

This chapter has been extracted from [AD4] with the written permission of Kayser-Threde GmbH. It gives an overview on the required functionality of MSL. The contents as taken from [AD4] has been reduced to an amount sufficient for understanding of the V&V activities.

This description represents the status **BEFORE** the V&V activities were started.

We thank the MSL project and especially Mr. Michael Birk for the support of this V&V activity.

## 2.1 General Overview

The MSL shall be tested, integrated and operated in three system configurations which are:

- the EM (Engineering Model) configuration
- the FM (Flight Model) configuration
- the SRM (System Reference Model) configuration.

The EM and the FM model represent the target configuration of hardware and software while the SRM uses representative hardware only. The software shall allow for execution on all three configurations without major changes.

From the software point of view the EM and FM are identical, the SRM application S/W will be different, but equivalent from a functional point of view, the differences are kept to a minimum.

The FM MSL is designed in order to operate in the MSRR in the US-Lab module of the International Space Station. It will be operated by a scientific team from ground or by the onboard crew by means of a Crew Terminal. The EM is identical with the FM from software and operational point of view.

## 2.2 Configuration and Functions

The general System Configuration of MSL is depicted in figure 2-1.

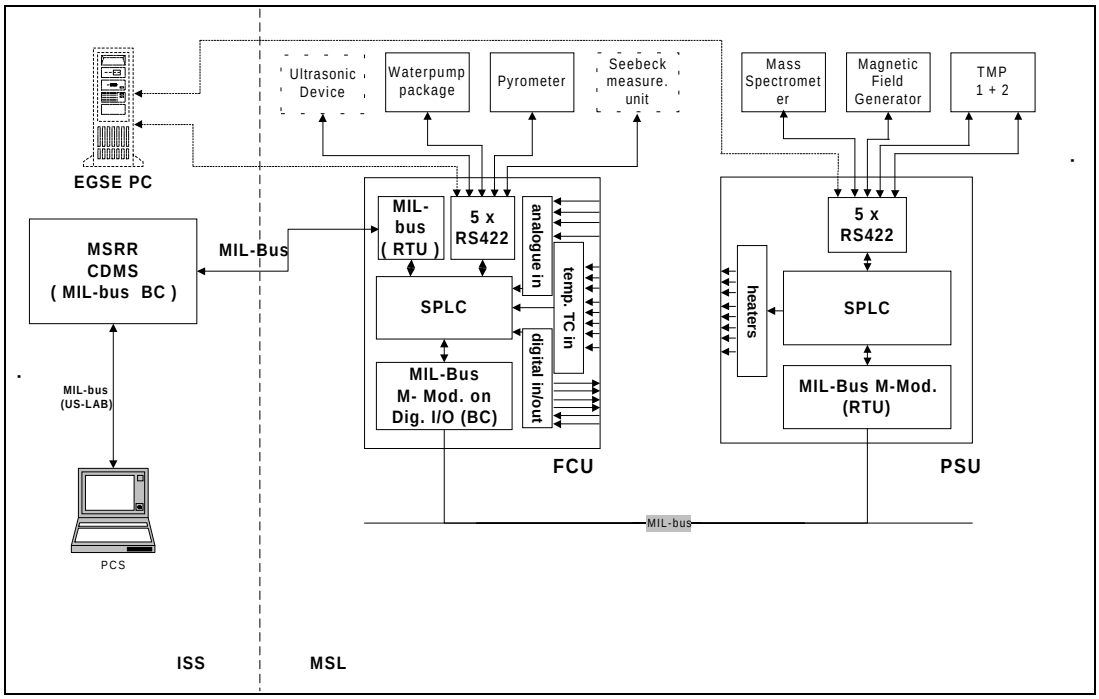

*Fig. 2-1 : Block diagram of MSL in EM / FM configuration.*

In EM/ FM configuration two SPLC computers (FCU and PSU ) and respective software are responsible for the functional and operational control of MSL respectively the components and subunits it is composed of.

The MSL S/W allows that its functionality can be distributed and can be allocated on the two SPLC units in the FCU and the PSU respectively and as well can run on a single computer unit in the Science Reference Model (SRM).

The MSL S/W provides the following functionality that s allocated on the FCU:

- The general main function of MSL S/W on the FCU is the overall control of the facility and of the control of the processing of a sample

  This main function is composed of a number of subfunctions.

  - Identification of furnace code and assignment of furnace parameters.
  - The loading of the sample into the furnace can be controlled. The sample is identified and an according set of parameters and an according sample processing sequence is selected.
  - The start of the sample processing is initiated by command but only if a validation of the sample parameters and of the facility status does indicate normal conditions.
  - The sequence of the sample processing and the survey of the respective parameters is controlled. The general sequence of processing steps is as follows:
    - exchange or loading of a sample
    - closure of the door
    - evacuation of the furnace
    - check of the sample integrity
    - heating up phase of the furnace comprising :
      - temperature control
      - limit checks
      - Peltier pulsing and Magnetic Field application
    - Quenching
    - cooling down phase
  - These control functions utilize the data acquisition and subsystem control functions.
    - The MSL S/W acquires the analogue and digital data - temperatures, pressures, status signals - and these are used to control the parameters and the sequence of the sample processing
    - and to control the following subsystems via RS422 interface link
      - mass spectrometer
      - magnetic field generator
      - turbo pump 1 and 2
      - pyrometer
      - water pump
    - and to provide a bi-directional RS 422 data link for:
      - ultrasonic diagnostics device (not installed but supported)
      - experiment dedicated electronics (e.g. Seebeck Measurement Unit; externally provided

- Control the following subsystems and components via digital I/O and analog input interface
  - Accelerometer
  - Core Facility
  - Current Source
  - Direct Current Converter
  - Gas Supply
  - Heater Current Source
  - Peltier Current Sources
  - Power Distribution Module
  - Power Supply Unit
  - Quench Drive Electronic
  - Sample identification
  - Sample Reservoir Heater
  - Stepper Moter
  - Vacuum/Gas Distribution Subsystem
  - Water Cooling System
  - Water Pump Package

- the commands that control the MSL S/W functions are uploaded to the MSL S/W by means of the MIL-STD 1553 B communication between MSRR Master Controller and FCU. The commands comprise a number of different types
  - the upload of sample processing parameters and the definition of the sample processing sequence
  - immediate control commands for interactive sample processing
  - commands for facility checkout
  - commands for data format configuration and data dump initiation
  - commands for MSL application S/W upload
  - commands for facility configuration and set-up
- the data downlink is performed by means of the MIL-STD 1553 B communication between MSRR Master Controller and FCU.
- In the case that data downlink is not possible due to e.g. LOS data are stored onboard the FCU.
- The MIL-STD 1553 B communication protocol is provided by the SPLC BSW package. It is developed and verified only once under SPLC contract and provided as generic function to the SPLC application S/W developers as part of the SPLC BSW package.
- the communication with the PSU is performed by means of a MIL-STD 1553 B communication. This communication is controlled by the FCU. The S/W provides the functionality of a MIL-STD 1553 B bus controller (BC)
- a supervisor function in the MSL S/W on the FCU is responsible for Failure Detection Isolation and Recovery (FDIR) functionality. This comprises :
  - the detection of critical situations by checking the acquired data and parameters against upper or lower limits
  - the inhibition of functions or actions if a conflicting situation may result e.g. switching on / of the turbo pump while the vacuum is not appropriate.
  - the initiation of actions to bring the system back to nominal or safe conditions as e.g. switching off the heaters and / or subsystems
  - driving the system into the safe mode
- the supervisor is a S/W function that runs on high priority. It controls the data and control flow from and to the data acquisition, the heaters and the subsystems.

- The operation modes are identified as:
  - stand-by mode - the purpose of this mode is to provide a defined safe and stable condition of the furnace and the subsystems. The door of the furnace is locked. The door is opened due to a user command to allow for sample exchange.
  - safe mode - The safe mode is entered whenever any form of inconsistencies and parameter limit violations are encountered. Being in this mode the furnace is driven to a safe condition by e.g. switching off the heaters. Ground and / or personnel is informed about the status and possibly critical situations and is prompted for commands in order to stabilize and normalize the situation. Safe mode is also entered when the command “abort” or “goto safe mode” is given.
  - processing mode - this operation mode allowsfor execution of the predefined and sample associated sample processing commands and actions sequence. It is possible to suspend and to resume the processing.
  - Test mode - this operation mode allows for testing and checking the furnace and the subsystems interactively by predefined sequences or by step-by-step commanding.
  - reprogramming mode - this operation mode allows for reprogramming of the MSL application S/W.

The MSL S/W provides the following functionality thatis allocated on the PSU :

- The general main function of MSL S/W on the PSU is the control of the power outlets for the heaters and subsystems
  - the MSL S/W running on the PSU SPLC processes the heater control algorithms. Each of the heaters is controlled by its dedicated algorithm. According to the results of the algorithm processing the power outlets for the heaters is controlled.
  - heater power is delivered on a number of power channels to which the consumers are assigned. The maximum consumable power on each of the channels is limited. The MSL S/W supervises and controls the total power consumption on each channel to which consumers are assigned.
  - the MSL S/W running on the PSU SPLC controls the Mass Spectrometer, the Magnetic Field Generator and the two turbopumps via RS422

The functionality provided by the S/W running on the PSU SPLC is under control of the MSL supervisor function allocated to the FCU S/W. This implies that the data exchanged between the two computer units via MIL-STD 1553 B bus communication includes data for function control purposes.

The software runs on a Standard Payload Computer (SPLC) a VME bus based Controller module. I/O modules provide analogue, digital and special purpose input output interfaces.

### 2.3 Software Model

The EM/FM software functions will be assigned to two SPLC computer units (the FCU and PSU) interconnected via an MSL internal MIL bus ( please refer also to Fig.2-1). The peripheral subsystems and the interfaces with them are specifically designed for MSL.

MSL and MSRR will communicate (telecommanding and telemetry) via a MIL 1553B bus. The MSRR will be the bus controller (BC) MSL will be a Remote Terminal (RT). The crew terminal itself is a RT. The MSRR is responsible to control the communication paths between MSL and the outer world. MSL will not be able to distinguish between commands originated by the ground or crew control . Telemetry

data is just sent to the MIL bus. MSL is not able to indicate a direction for the data (eg. either Crew terminal or ground control) MSRR is responsible to distribute the data according to the requests of the different users.

The Science Reference Model (SRM) is operated on ground and serves as equipment to develop and test scientific experiments prior to execute them in the FM in orbit. It is functionally and operationally compatible to the EM/FM.. But it s composed of commercial industrial components as far as possible to save expenses without being obliged to develop new software. The design guideline for the SRM configuration is to keep the hardware costs and the software efforts and costs due to changed hardware components at a minimum.

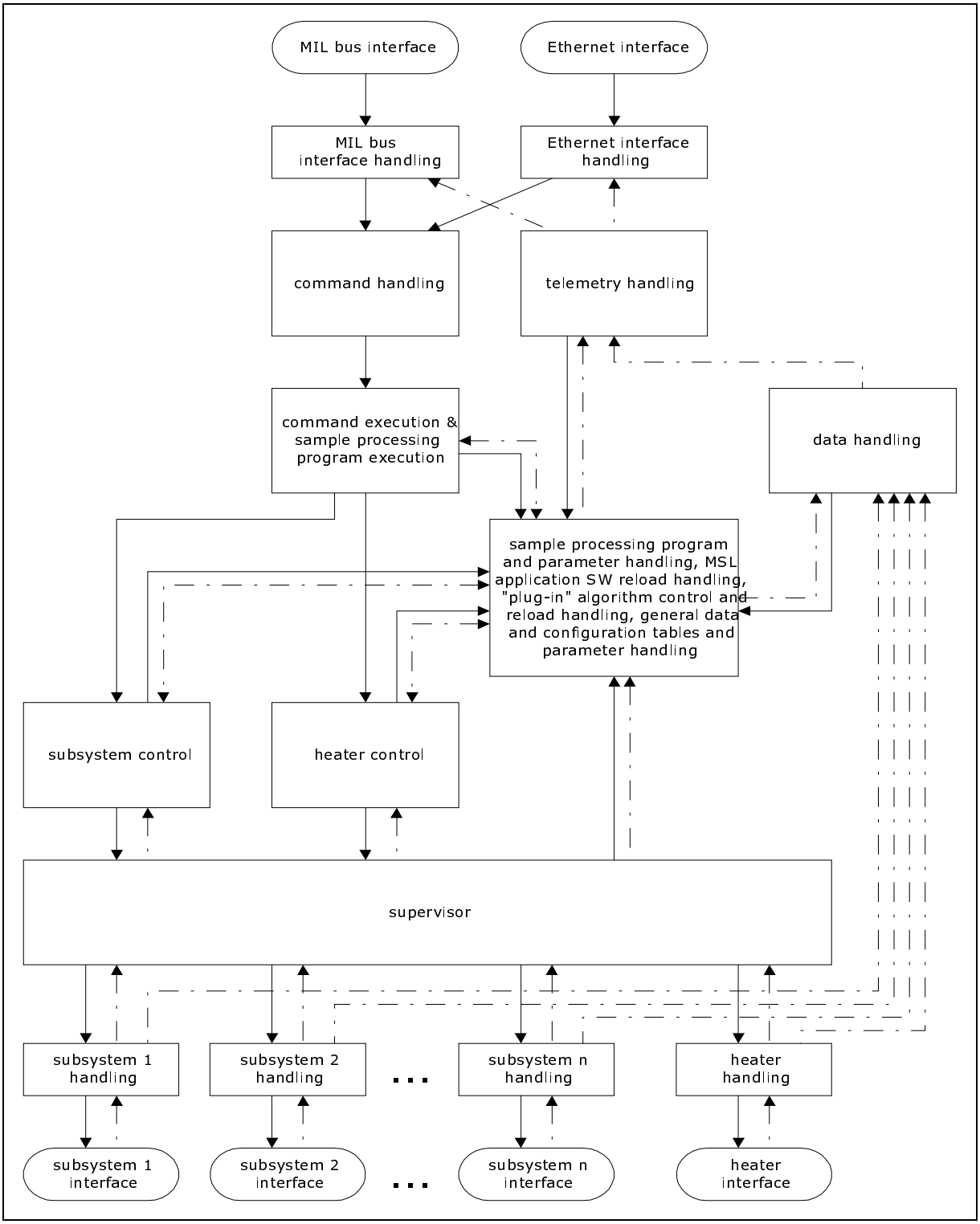


*Fig. 2-1: Overview of the MSL Software Model*

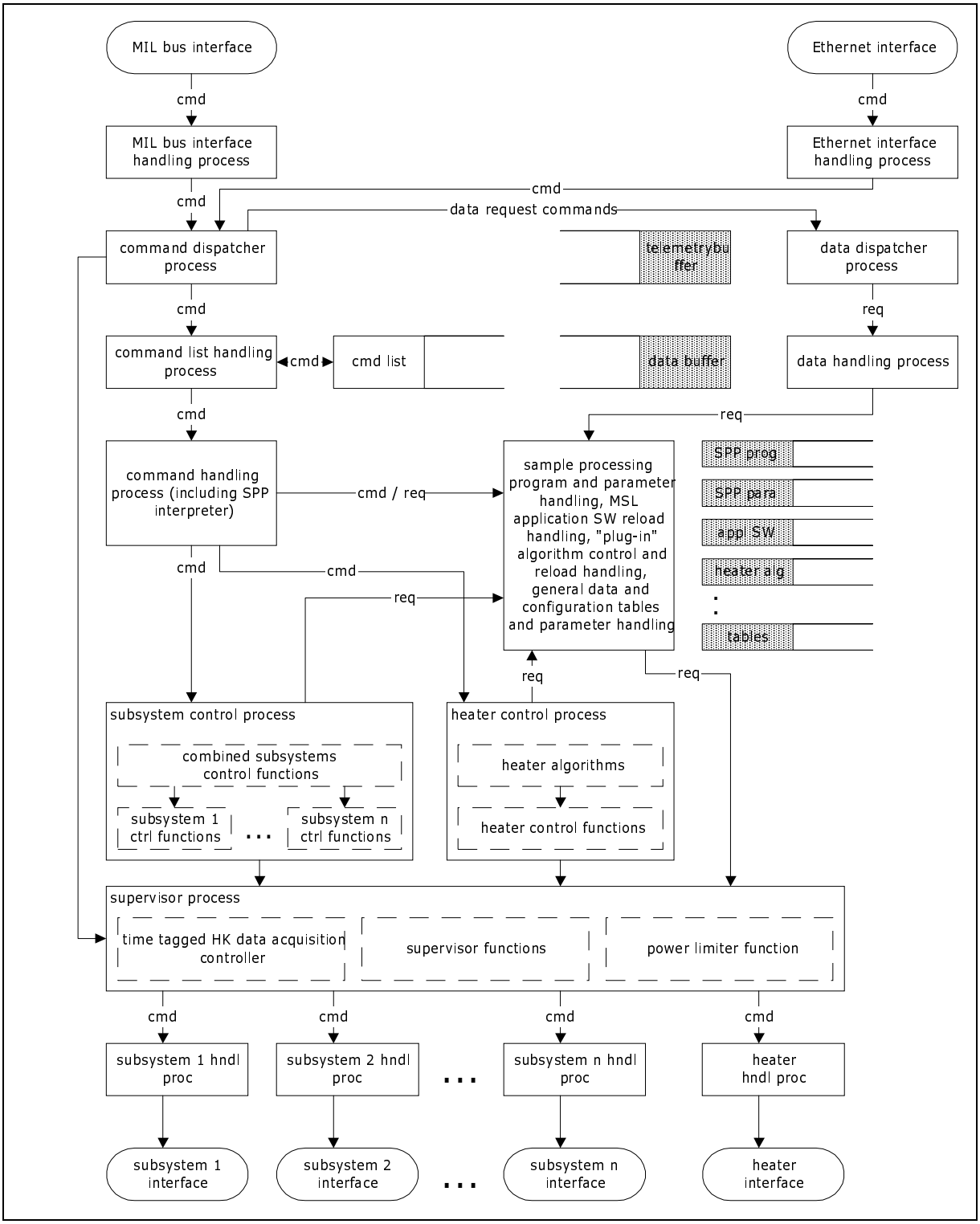


*Fig. 2-2: Control Flow in the MSL Software Model*

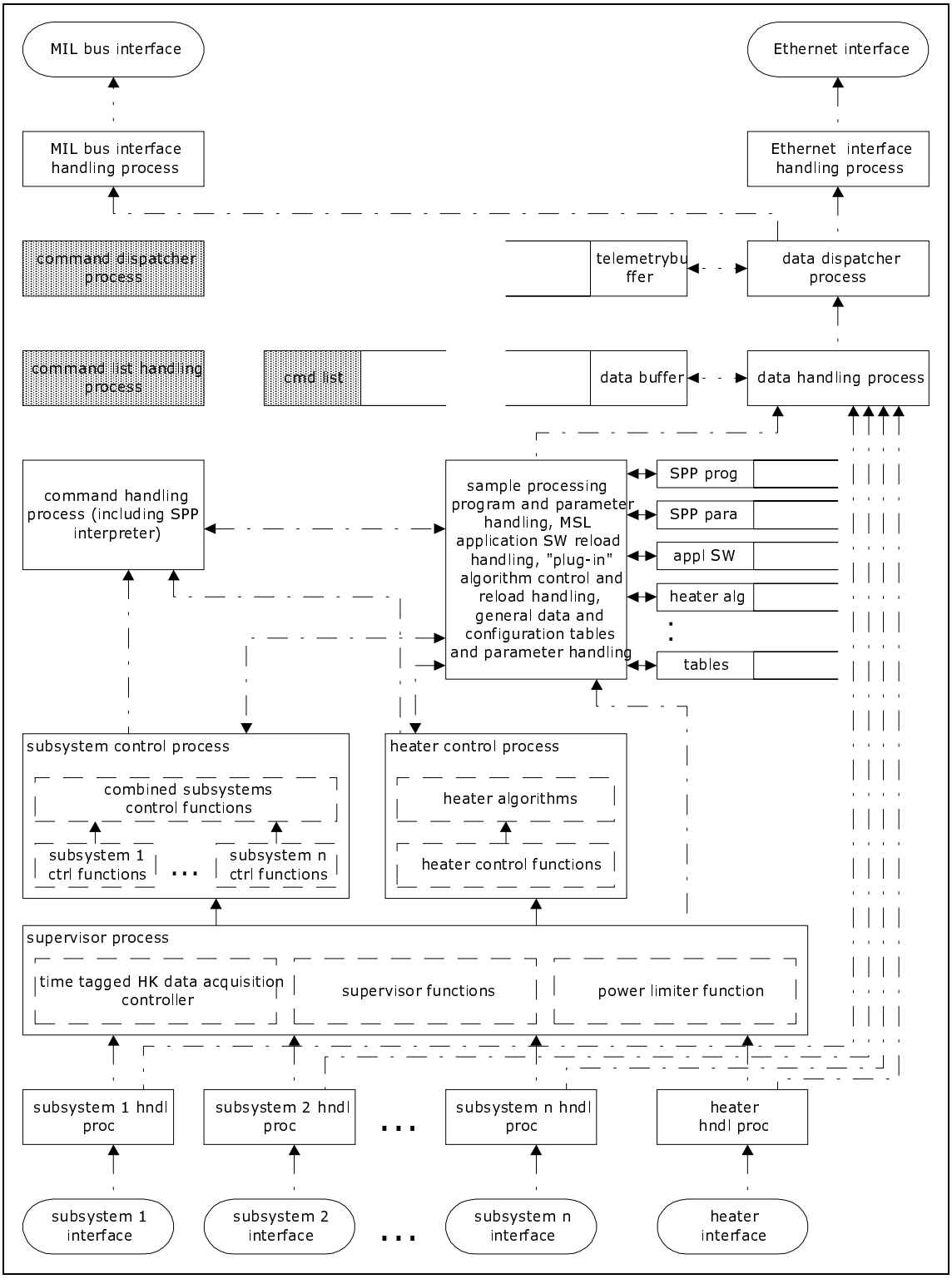


*Fig. 2-3: Data Flow in the MSL Software Model*

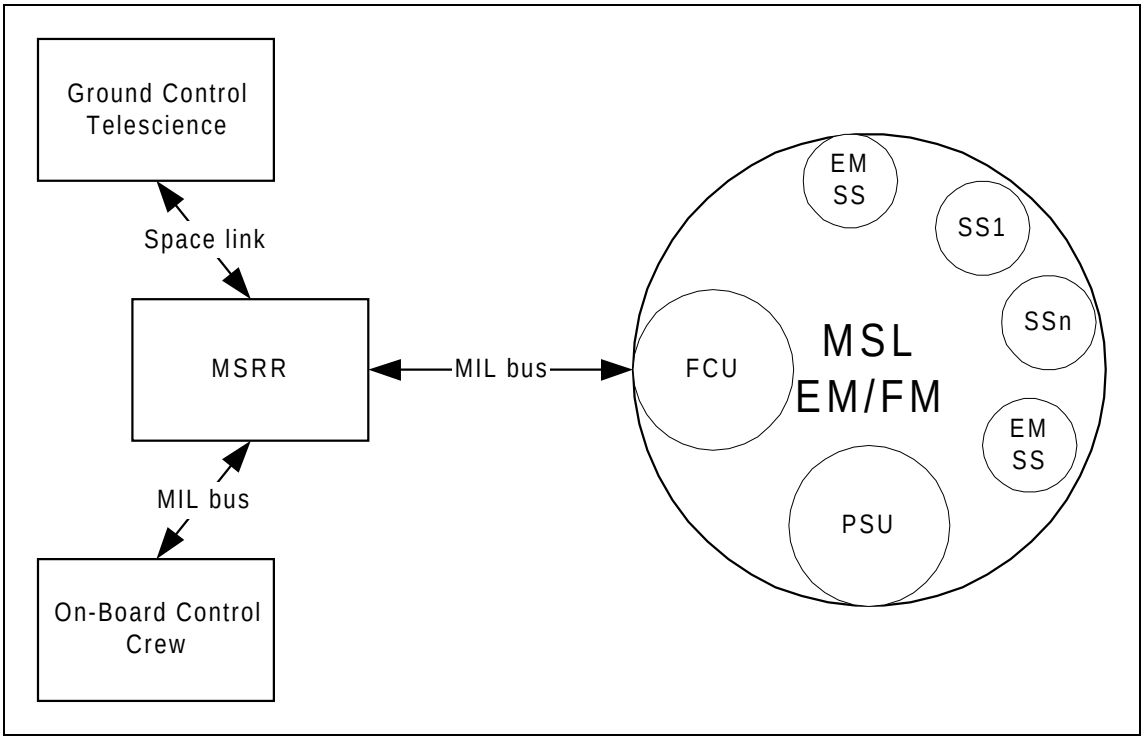


*Fig. 2-4: Context Model of MSL in EM/FM configuration*

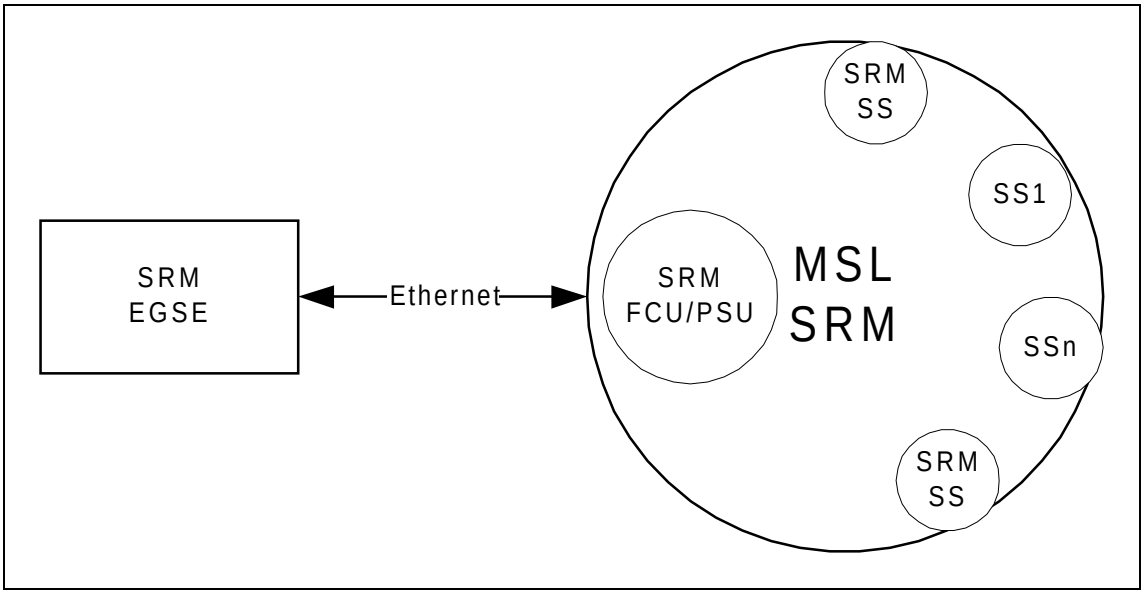


*Fig. 2-5: Context Model of MSL in SRM configuration*

In the SRM configuration the software runs on one computer unit instead of two that is software compatible to the SPLC ( SPARC V7 computers). Compared to the EM/FM model the configuration of the peripheral subsystems is different ( e,g, the heater current supply are commercial power supply units controlled via GPIB interface)

The design of the MSL software copes with the requirements of the different MSL models.

The real time operating system VxWorks from Windriver Systems is used for MSL.

The SPLC Software Development Environment consists of a Workstation and a ETHERNET connected SPLC target computer. For the software development and the downloading to the target the TORNADO development tool from Windriver Systems will be used.

## 2.4 Control and Data Flow

The following figures give an overview on the principal command flow, the supervision of control and data flow and the startup control flow.

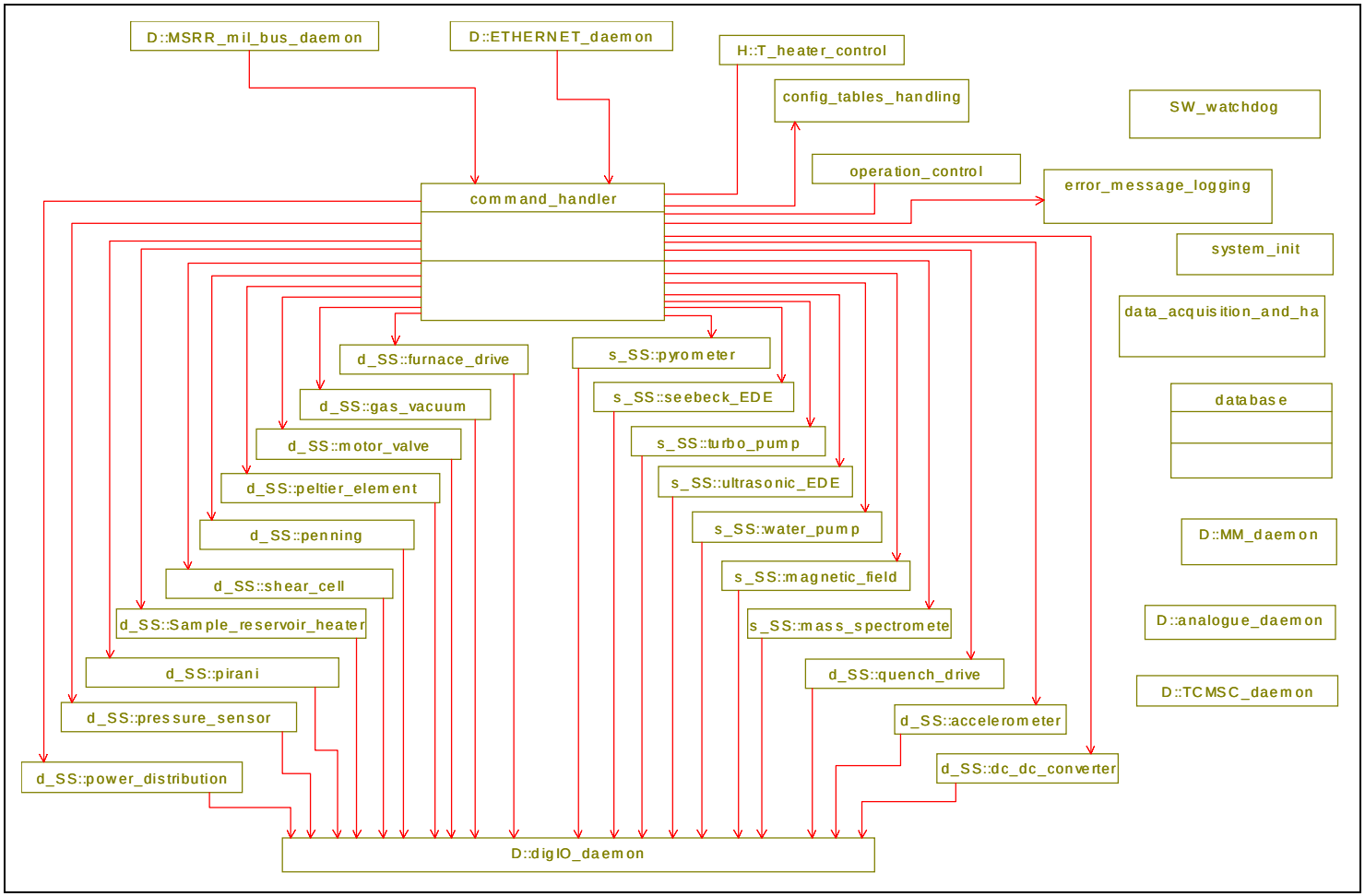


Fig. 2-6: Overview on Command Flow

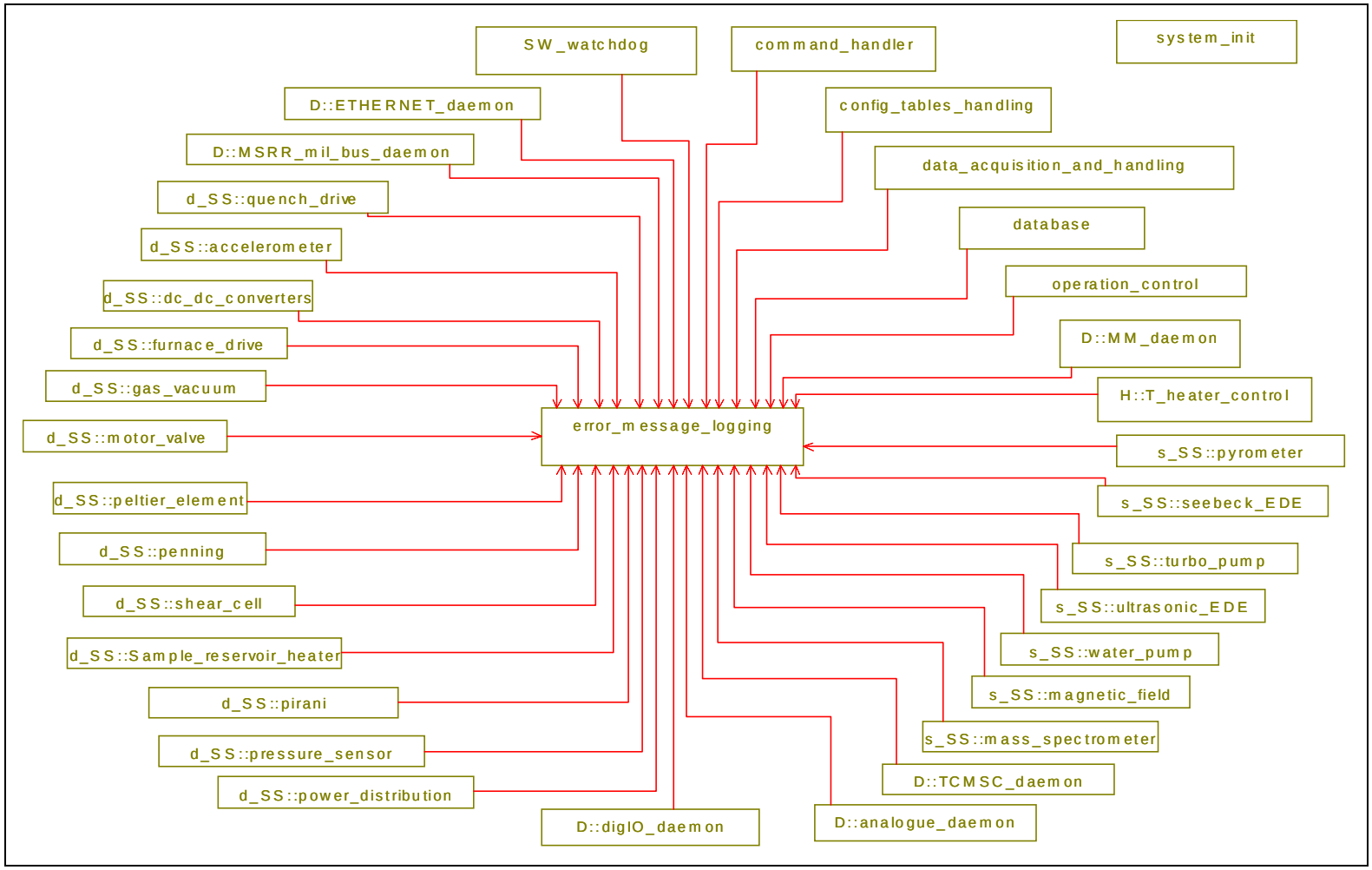


*Fig. 2-7: Supervision of Control and Data Flow*

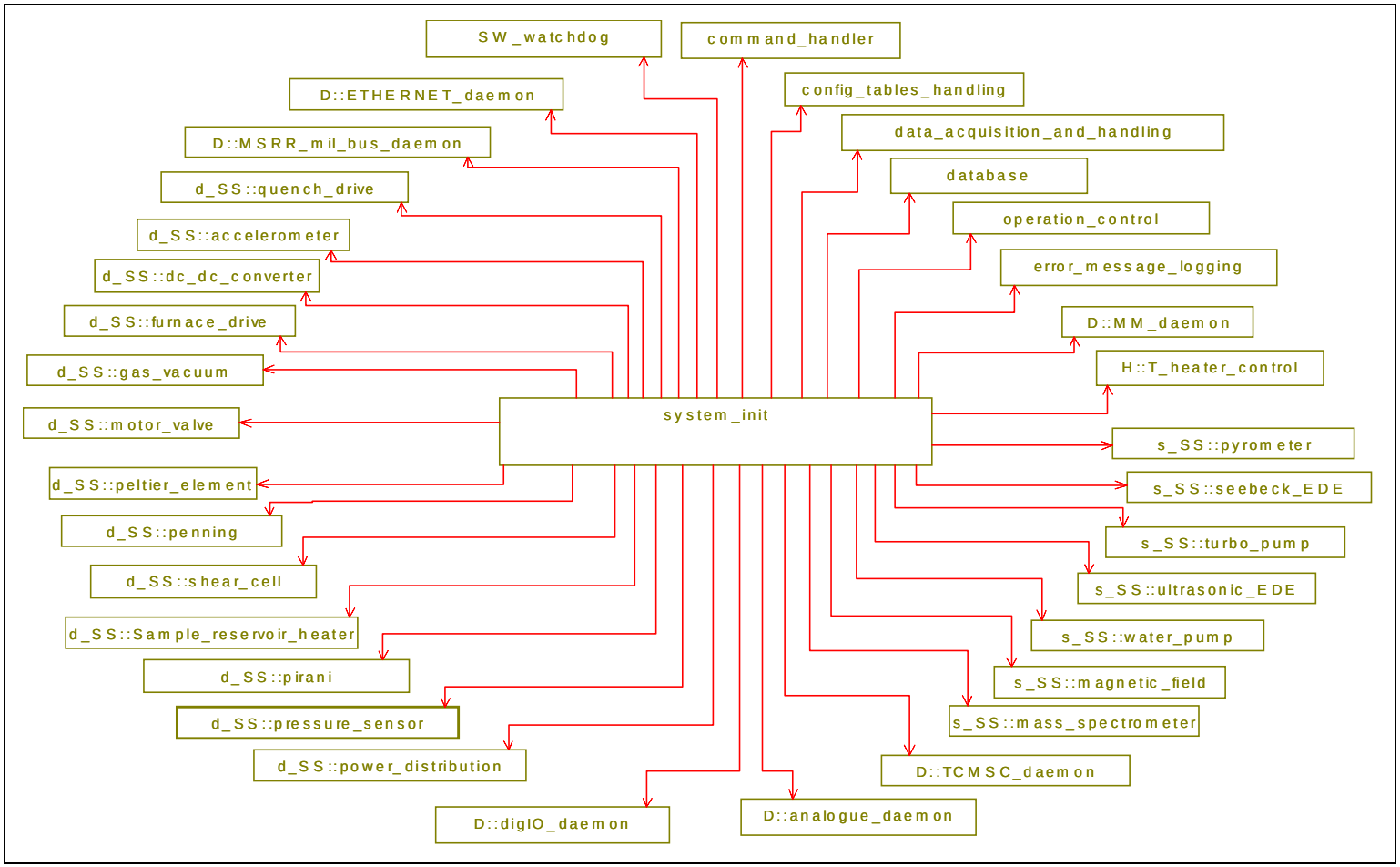


Fig. 2-8: Startup Control Flow

## 2.5 Standardization of Intertask Communication

The general problems related to the MSL model require a strongly modular design.

It shall be possible to run any task / process that is independent of hardware interfaces on either CPU (FCU or PSU) or on a single one (SRM).

The task intercommunication is preferably realized by the UDP/IP protocol. It shall be proven by early system verification that this protocol is compatible with the SPLC´s performance. If not a VxWorks provided standard message handling shall be implemented.

The message format for intertask communication shall be standardized by a generic format :

```
typedef struct {
        TyCmd           cmd;
        TyPriority      prio;
        TyDev           source_task;
        TyInstRg        source_task_instance;
        TyDev           dest_site;
        TyDev           dest_task;
        TyInstRg        source_task_instance;
        int             length_of_attached_data;
        int             ptr_to_attached_data;
        int             type_of_attached_data;
        int             info;
        TyTime          start_time;
        TyTime          act_time;
} MSL_message_header;
```

The attached data are added to the data stream of the fixed-size header for transfer. They are stored into a (random-access) buffer when received by a process.

On reception of a message a task analyzes the message header and executes it according to the contents fofthe command (cmd) field:

```
switch cmd

{
        case type xyz;

        case type abc;

        default
                ILLEGAL_TYPE_ID;
}
```

When Using UDP /IP for intertask communication, the software interface at the internal MIL bus link between FCU and PSU shall be UDP/IP compatible.

## 2.6 Principal Components of the Software

The functionality of the principal MSL software components is briefly described by the following sections.

### 2.6.1 Command Handler

The command handling process is one of the central processes of the MSL software and comprises:

- command dispatching
- command control

The command handler receives commands from external and internal interfaces (e.g. operation_control and heater_control) performs checks and preprocessing and controls the execution. The commands that are allowed to be executed depend on the operation modes:

- standby mode
- normal operation mode
- self test mode
- reprogramming mode
- test mode
- checkout mode (*not* for in-flight operation)

and the status of the system or particular subsystems.

The command handler software object is designed to provide generic functionality including the capability to execute userdefined command related functionality.

The command handler receives the commands coming from one of the following command sources:

- the Ethernet interface
- the MIL-STD-1553-B bus interface
- MSL software processes (e.g. operation_control and heater_control)

The commands are processed as follows:

1. formal validity check on CRC and syntax
2. interpretation of the command ID
3. check of the access identification / password
4. check of the permission to be executed
5. handing over to the destination process.

Each command is acknowledged by the destination task to the source task when it is accepted for execution. All commands are checked at the hierarchical highest software level possible. If a command is rejected the reason for rejection is added to the message returned from the source task.

The hierarchical highest level w.r.t. the operator on ground or on board is the command dispatcher itself. It checks the command on validity and permission to be executed and returns an acknowledgement or rejection with indication of the reason.

Time tagged commands are handed over to the operation control process for postponed execution.

All possible commands are defined in [RD01] in the format of a MS-ACCESS database. With respective support software command tables and handling procedures for MSL flight and ground software are generated from this database.

#### 2.6.1.1 Command formats

A command has the following general structure and is transported in the standard message container:

| Command ID | destination | Command parameters | timetag | Operator ID | Operator PIN |
|---|---|---|---|---|---|

The command ID is unique for each possible command. It is generated from the above mentioned command database and the respective support software.

For most of the commands the command destination is strongly associated with the Command ID. For this type of commands it is possible to perform a table driven dispatching of the commands:

command_ID → dest_task;

For these commands the destination field normally is 0 and the destination is determined by means of the dispatcher table. If, for user defined reasons, the destination is not 0 the destination in the received command is used. This is especially valid for the ”transparent commands”.

The transparent commands are created to allow the MSL system user to have direct access to a particular subsystem. The content of the command is not interpreted by the MSL software but is handed over directly to the subsystem interface. A possible data return of the subsystem is passed back to the user in the same transparent way.

The timetag in all commands defines whether a command is executed immediately ( timetag = 0 ) or at the distinct absolute time ( timetag > 0 )

#### 2.6.1.2 Dispatching Handling

The command dispatching is controlled by a table, the dispatcher table comprising the following entries with associated functions :

- syntax check
- preprocessing
- command supervisor

The command dispatcher calls the functions pointed to by the table sequentially passing the command as actual parameter. If a function pointer is NULL the action is skipped. According to the returned value the command processing continues or is interrupted and a negative acknowledge is given.

The **syntax check function** contains the algorithm to evaluate the correct and valid syntax of a particular command. The design allows to establish a unique procedure for each command.

The **preprocessing function** allows to preprocess a command and its parameters e.g. convert parameters or change parameter settings according to actual system parameters.

The **command supervisor function** checks whether the command is allowed to be executed e.g. depending on the actual system status and/or particular HK data.

All other SW processes access the dispatcher table by an interface routine (get_command_characteristics() ) in order to evaluate the executability of a command that is generated by them. As the table has information on all available commands it is capable of checking all commands and rejecting those commands that would lead to conflicting situations.

In case a command is identified as forbidden to be executed the operator ID and Operator PIN is checked. An algorithm based on CRC calculation (the definition of the algorithm can be found in the detailed design part in Annex A) calculates a unique PIN from the Operator ID. Each Operator that is allowed to issue commands to the MSL in off-nominal conditions is provided with an ID and an associated unique PIN. The MSL SW passes over a command for execution despite the prohibition if the ID and the PIN associated with the command are matching. SW generated commands do not use the ID/PIN capabilities.

Before handing over the command to the destination a command sequence ID is assigned to the command. This ID is used to uniquely identify a particular command in the general commanding sequence. This ID is generated by incrementing the ID variable (modulo 32bit).

Each command together with its execution status is handed over to the central message logger.

#### 2.6.1.3 Command control

A command that was preprocessed correctly is passed to the destination and if required (defined by the control_execution flag) its execution is controlled. For this function a data structure is defined into which the command is posted by the command handler. This structure is used to survey the execution of the command by cyclically checking whether commands are already timed out. Each command receiver is obliged to return a command acknowledgement or a rejection in conjunction with a reason for rejection. The command handler passes either the returned error message from the command receiver or its own generated message due to a failed acknowledge to the central message logger.

Each command is acknowledged after its execution, either successful or with failure indication. Commands that take rather long to be executed are ”pre”acknowledged when their execution is initiated.

With the execution acknowledgement – not the ”pre”acknowledgement - of a particular command it is released from the command control function.

### 2.6.2 Data Acquisition and Data Handling Processes

The Central Data Acquisition and data handling processes consists of the following main functional elements :

- Data handling function
- Data Dispatcher Function
- Data Monitoring and Data Supervision Function

#### 2.6.2.1 Data Handling Function

The data handling function controls the data flow through the system and controls the database. It receives all science and housekeeping and status data directly from the hardware interface daemon processes, and other software functions and stores them in the database. Possible data sources are :

- Analog interfaces handled by the analog interface daemons (interrupt service routines)
- digital I/O interface handled by the dig I/O interface daemons
- serial line daemons
- all other processes /tasks generating status data or secondary data by processing primary acquired data.

The data generating functions pass the data either as single entities or as data blocks. The interrupt driven data sources are:

- analogue input
- digital input
- serial data input

They own their respective data input buffer at the data handling function. To this buffer the interrupt service routine copies the data directly from the hardware. The actually delivered data are identified by a buffer ID determining the content of the buffer. The data source issues a message to the data handling function to signal the data delivery.

The other data generating tasks deliver their data as part of the message in the standard message header.

From the buffer or the input message the data are copied to the database. When performing the copying for each data item the following processing steps are performed:

For each data item a correction, calibration, or conversion function is defined in the database. As part of the copying process the data handling function checks the according entries in the database and if the entry is not NULL manipulates the data using the respective function before storing them in the database.

Requests on data entities by other SW functions (e.g. data dispatcher or subsystem control functions ) are fulfilled by providing the data either as single item or as block.

The data, either single items or as block are identified by a data ID. This ID is unique and is derived from the above mentioned I/O signals database. The data handling function provides call interfaces to get access to the data ( put_data, get_data).

#### 2.6.2.2 Data Supervisor Function

The data supervisor is responsible for checking limits on the housekeeping data values and combinations of housekeeping and/or science data values to avoid possibly critical situations in the system and the experiment. It is part of the data handling function.

For each data item a supervision function can be defined in the database. When putting a data item into the buffer the data supervisor function checks the according entries in the database and if the entry is not NULL hands the data over to the respective function.

The processing of the supervision function results in the generation of a message to the central error and message logging function if the supervision criteria are violated. The return value of the called supervision function ( if < OK) is handed over to the message logging function.

### 2.6.3 Telemetry Dispatcher

Cyclically or on request, data are formatted into telemetry data frames and handed over to the output interfaces. The definition on the required contents of the telemetry frames is kept in respective tables. The data dispatcher requests data from the database according to the contents definition, creates a telemetry buffer and stores the data in the buffer in a sequence according to the contents definition. When filled up it is handed over to the output interface daemon process which is either the MIL bus interface or the Ethernet interface as appropriate on request.

### 2.6.4 Error and Message Logging Function

This process serves as a central SW anomalies handler. All contingencies detected by the software passed to this message and logging server. It is responsible to receive messages in standard format from all other processes / tasks comprising the MSL software. Messages stored to a respective file in the mass memory and downloaded on request or immediately downlinked according to the setting of the respective system parameter.

The message logging process is running as an instance on the FCU and on the PSU. Each of the instances performs autonomous message supervision and initiation of possible reactions. The instance running on the PSU reports to the instance on the FCU. The instance on the FCU performs the storage and the downlink. In SRM configuration only one instance for the complete system will be running.

### 2.6.5 Interface Handling Processes

#### 2.6.5.1 ETHERNET Daemon Process

The Ethernet interface handling process provides the hardware/software interface to the Ethernet bus. This interface will be inactive in the MSL US LAB installation and configuration but will be the system I/O interface in SRM configuration. The Ethernet Daemon process will provide a standardized I/O interface using the standard message container that is on peer level with the MIL bus interface.

#### 2.6.5.2 MIL Bus Interface Daemon Process

The MIL bus interface handling process provides the hardware/software interface to the active MIL bus with the MSRR. It receives all commands coming in from on the MIL bus and hands them on to the command dispatcher process. In the opposite direction, it receives data from the data dispatcher process and interfaces them to the MIL bus.

The MIL bus Daemon process will provide a standardized I/O interface using the standard message container that is on peer level with the Ethernet interface.

#### 2.6.5.3 Digital I/O Daemon Process

The Digital I/O daemon process provides a standardized software I/O interface using the standard message container with the Digital I/O hardware. For each of the hardware interface an instance of this process will be installed (see (RD01])

#### 2.6.5.4 Serial I/O Daemon Processes

The serial I/O daemon process provides a standardized software I/O interface using the standard message container with the serial RS422 interfaces. For each of the serial interfaces an instance of this process will be installed. (see (RD01])

#### 2.6.5.5 Analog Input Daemon Processes

The Analog Input daemon process provides a standardized software I/O interface with the Analog data acquisition hardware. For each of the hardware interface an instance of this process will be installed . (see (RD01])

#### 2.2.6.6 Mass Memory Daemon Process

The mass memory daemon process provides a standard process interface with the file handling and management system of the BSW.

# 3. Introduction to ISG

This chapter introduces into the basic concepts of Instantaneous System and Software Generation (ISG). It includes information extracted from the ISG User's Manual [RD12]. ISG forms the base for the formal definition of the MSL software and its verification and validation.

ISG is based on the formal specification of behaviour and the idea of early system validation initiated by ESTEC [RD3-RD4]. It extends and optimises the basic ideas and provides an efficient solution for complex applications.

## 3.1 The Idea and Its Realisation

### 3.1.1 Overview

The principal idea (Fig. 3-1) is to provide a facility which allows

- to generate a system or program from a minimum of engineering information,
- to automate the generation of an executable system and to provide the environments for simulation and the target system within minutes, and
- to provide an immediate feedback from the executing system by graphical figures and reports.

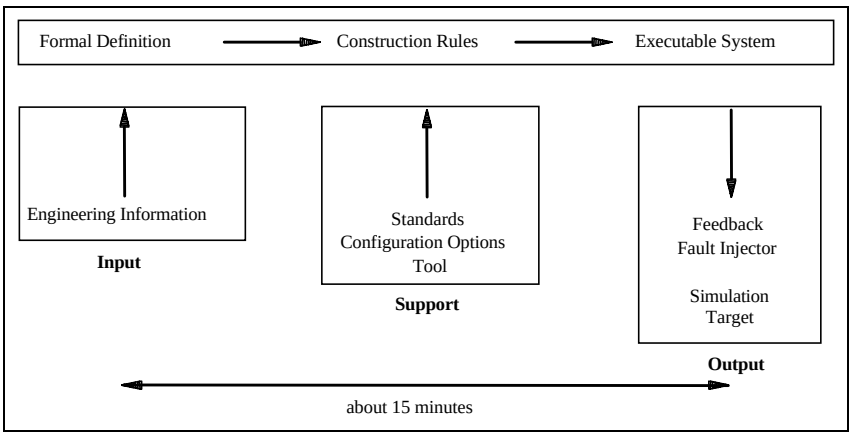


*Fig. 3-1: Automated and Instantaneous Construction of a System from Engineering Information*

Usually, it is very time consuming to implement the architecture of a system and it takes a lot of time until the first feedback is available. To shorten this time the idea of "Instantaneous System Generation" was borne.

ISG takes the information about the architecture and topology, the behaviour as expressed by "input-processing-output", the performance figures and constraints and converts them directly into an executable program which can be configured either for simulation or for execution on the target system. ISG allows a system engineer to immediately obtain a feedback on his (first) ideas and to perform iterations until a satisfying solution is achieved. The functional refinement of the system can be done in an incremental manner by plugging in the application functions into the behavioural skeleton established by ISG.

The generation of the executable by ISG and a run usually takes about 15 to 20 minutes depending on the complexity and the size of the program. In case of MSL such a run for generation of an executable takes

about 40 minutes. Its execution until the desired coverage is achieved takes about 20 minutes at a command injection rate of 1/s.

### 3.1.2 The Organisation

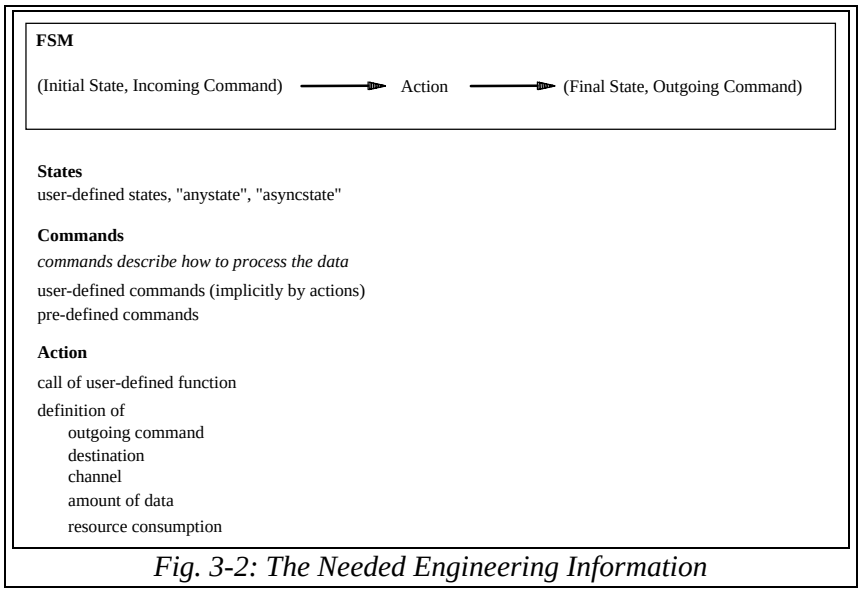


*Fig. 3-2: The Needed Engineering Information*

ISG takes a formal specification of behaviour, performance, topology and communication based on Finite State Machines which is delivered by the engineer by a table (Fig. 3-2). However, a user is not forced to define "big" state machines, if desired it is sufficient to define just one state. Additionally, generation and configuration options have to be provided by two more files. The ISG toolset takes this information and builts automatically a program.
For automated (stress) testing the fault injection feature is available which will insert additional code or modify the code in order to inject faults into the system during execution.

As far as detailed functional algorithms are missing ISG provides decision criteria for execution of the program logics in order to allow for automated testing. Actions as defined by the engineer are either selected randomly or sequentially one after the other. In latter case a repetition factor can be given so that a group of actions may be repeated before the next group is executed.

### 3.1.3 Feedback and Visualisation

As feedback the program generates reports, data flow and timing diagrams.

Reports inform about coverage of executed actions, states and state transitions, execution time, consumption of resources, injected faults and observed exceptions like a time-out or a cycle overrun. The delivered data may also be displayed by a spreadsheet program.

For tracking of data flow "Message Sequence Charts" (MSC) are produced (see chapter 7), but on request any other format can be supported as well. The program may generate an MSC file in a format compliant with the Z.100 standard of ITU [RD11]. The contents of such a file can be displayed by MSC editors like the one from ObjectGEODE [RD09] or Telelogic/SDT [RD10] and might be taken for further verification as supported by SDL toolsets.

However, the data flow can also be displayed on a screen by the ISG MSC viewer.

To allow for a better understanding of what is going on ISG provides some extensions not included in the ITU standard. Firstly, it allows to format and select information from what is flowing through the communication channels: a user does not need to define data exchange formats in an artificial manner to get the desired information displayed in a MSC. Secondly, the communication channel may be printed with the data to visualise the transfer medium. Thirdly, a user may print debugging information like which data are coming into a function or are generated by a function, when and where a state transition occurs, a fault is injected or identified, or a timer expired.

Timing diagrams (see chapter 7) allow to track the data flow over time at certain locations which may be processes or devices like buses. In interactive mode a user may click on a certain event and the contents of the message will be displayed (see chapter 7). By filter criteria a user may select a certain subset of the data flow.

### 3.1.4 Real-Time Processing

The ISG toolset supports real-time processing. Timer events and timeouts can be created and be reset, cyclic tasks are supported and their completion within a given deadline can be monitored.

### 3.1.5 Fault Injection

Fault injection is supported by: (1) injection of erroneous commands / data, and (2) loss of signals / message. Fault injection may either be done explicitly by a user or automatically by the toolset.

### 3.1.6 Integration with Existing Software

The ISG toolset provides the capability for integration of existing software. This may happen in twofold manner: (1) by linking this software directly with the ISG/V generated software, or (2) by remote access. In case of event-driven simulation it is possible to synchronise all executables.

## 3.2 The Principal Elements of the Organisation

The principal elements of the logical organisation are processes or (hardware) devices[1], states, commands, communication channels and resources. This chapter gives a definition of such principal elements.

### 3.2.1 Processes and Devices

A process or device is a unit which can perform actions in response to an input and which communicates with other such units through communication channels. When executing it consumes a resource which implies blocking of this resource for other processes and a delay until the next action can be performed.

A process usually executes on a resource like a CPU, a device may be a bus and then it consumes the bus cycles.

Processes and devices are the basic elements of a system architecture. The set of processes and devices consist of user-defined processes/devices and pre-defined system devices. A user is free to define his own set of processes and devices, but they must not conflict with the system-defined literals. Preferably a user shall define the literals implicitly by the formal specification of behaviour and performance which is provided by a table. Explicitly, a user can define literals by the file "easysystem.def". This is also true for states, commands, logical and physical channels and resources.

### 3.2.2 States

A process or device may take a number of states like depowered, powered, initialised or operational. Each state imposes some constraints on a process by limiting the processing capabilities: in state

[1] The terms "process" and "device" are synonyms of each other; for historical reasons the term "device" is more frequently used.

"depowered" a process is not available at all, in state "initialised" it may only perform a reduced set of actions, while in state "operational" its capabilities are fully available.

States allow to give a process different shapes.

Three sets of states exist: "anystate" (one member only), "asyncstate" (one member only) and "the set of user-defined states". This classification is of importance for handling of commands.

"anystate" and "asyncstate" are common states which allow processing of commands independently of what is going on in a FSM: Consequently, a process can run two independent actions at a time. it can execute a FSM step by step, receiving a command for each step, and in addition it can execute commands independent of the FSM's without disturbing execution of a FSM.

### 3.2.3 Data Processing and Commanding

The message format includes information ("command") about what the receiver shall do on reception of the message. A sender may advise the receiver to execute some actions on reception of a message only, or the command specifies which data are included and how to process them.

This allows to introduce a standard structure for a module like a process. Such a module identifies the incoming comands and then branches to the related processing sequence according to its state. Some of the commands may be common to all applications like to open or close communication lines. Some others may be specific. In case of an incremental development approach commands and states may be added step by step..

Hence, the principal structure will be based on

(1) a case statement regarding the states, and per state

(2) a case which covers all the incoming or possibly internally generated commands. Each branch of this case includes the corresponding actions.

This allows to provide (1) templates which can be reused by every application and (2) to apply fixed rules on how to expand the template to make it specific for an application.

Incoming commands are related to a state. As mentioned in the previous section states form three sets and commands are related to each such set:

- commands related to "anystate"
- commands related to "asyncstate"
- commands related to user-defined states.

When receiving a command related to "anystate" a separate brach is entered where actions are defined for all the commands which are independent of FSM's of a process. For "asyncstate" it is similar: commands related to "asysncstate" are processed immediately when a message is received from a physical channel. This is like interrupt processing and tehrefore processing should not take a long time in this case.

Hence, commands related to "asyncstate" will be served immediately by interrupting execution of commands related to "anystate" or "user-defined states". Latter two states will only be interrupted and suspended on action level (see structuring of the command procedure table in chapter 4).

As specific actions following terms are reserved:

- period starts a timer cyclically
- resetperiod terminates a cyclic timer

- timeout starts a timer to set a timeout for an expected command
- resettimer cancels a timer started by timeout
- deadline allows to check whether a deadline is met or not
- injerror initiates injection of an eroneous command (including suppression of a command).

### 3.2.4 Channels

For communication between modules like processes "logical" channels are defined which are mapped onto a set of physical channels like buses, RS232, RS422, TCP/IP, IPC, UDP, files, screens etc.

A number of physical channels may be grouped together to form a logical channel. Hence, two equivalent physical channels like two buses may form a redundant pair of channels. Arbitrary combinations are allowed. Redundant channels are identified by adding grouping information to the standard information about physical channels.

A standard transfer procedure takes the message and forwards it through all the physical channels selectively to the destination or in broadcasting mode. This approach decouples the application specific routines from the system's topology and allows to introduce generic subroutines for data distribution which are driven by data. This results in a high degree of reuse of the communication functions.

The message formats may also be used for module-internal communication, e.g. to be passed from function to function as parameter. This way it is very easy to switch from internal to external communication because no interfaces need to be changed. This prevents that changes of the topology or of software-hardware partitioning will seriously impact the implemented software.

The standardised format also allows to introduce a single interface to receive messages within a process. Similar to the output procedure the receiving functions convert the (physical) input into the same logical format. This allows to start processing of data or of other events from a single location within an application process.

To combine a number of physical channels to a set of redundant channels the "coverage" is introduced, which indicates that the data shall be transferred only through one of a set of channels. The coverage modes form the type "TyCoverage". Pre-defined modes are "allroutes", "allroutesreset", "onceonly" and "onceonlyreset". "onceonly" has to be added as attribute to a set of redundant channels to indicate that the data shall be transferred through the first available physical channel. If more than one redundant group shall be defined the postfix "reset" must be added to indicate the start of the next redundant group. The attribute "allroutes" indicates that this physical channel shall be selected anyway and independent of what happened previously for a logical channel.

The mapping between a logical channel onto a number of physical channels is defined by a file requiring the following format:

@[2] <dev> <logical channel> <physical channel> <coverage mode>

An example of such mapping is:

```
@    mms   ch1   bus11      onceonly
@    mms   ch1   bus12      onceonly
```

[2] A "@" at the first position of a line always indicates that this line is a valid, non-comment line. All other lines are considered as comment lines.

| | | | | |
|---|---|---|---|---|
| @ | mms | ch1 | bus21 | onceonlyreset |
| @ | mms | ch1 | bus22 | onceonly |
| @ | mms | ch1 | file1 | onceonly |
| @ | mms | ch1 | lan1 | allroutes |
| @ | mms | ch1 | file2 | allroutes |

This information instructs the system to perform following actions in case process "mms" issues a message through logical channel "ch1":

1. from a routing point of view:
   - there are four physical destination channels: (bus11/bus12), (bus21/bus22/file1), lan1 and file2
2. from a transmission point of view:
   - "onceonly" for bus11 and bus12 indicates that the message shall be transferred through the first physical channel which is available: hence, the message is transferred via bus11 if bus11 is available, if bus11 is not available it is transferred via bus12, if bus12 is not available the message is lost.
   - similarly, for bus21/bus22/file1 the message is transferred via bus21 or bus22 depending on which channel is available, and if both are not available it is saved to file1 (provided the medium is available).

     The mode "onceonlyreset" advises the system to start with a new group of redundant physical channels, and to forget that the message may have already been transferred via bus11/bus12.
   - lan1 / allroutes defines that the message shall be transferred via lan1 anyway
   - the same is true for file2/allroutes

Hence, "allroutes" initiates a transfer in any case, "onceonly" limits a transfer only to one physical channel out of a group of redundant (physical) channels. The postfix "reset" enforces a reset of the coverage counter which counts the number of transfers for a redundant group of physical channels.

Generic rules for the mapping from logical to physical channels can be given by the file "easysystem.def".

If a logical channel is mapped to one physical channel only and a user does not want to have an option at run-time to add another physical channel, then by a configuration option code can be configured such that the physical channel can directly be addressed without access of the mapping file.

### 3.2.5 Resources

Resources are the hardware elements of the system like CPU, buses or other physical channels. For simulation or linking it must be known on which resource a process executes and which resource it consumes.

For a formal specification it is needed to specify the resource on which a process or device resides.

Consumption of resources is expressed by the number of resource cycles which are consumed. A range and a mean value of resource consumption have to be specified for simulation. This information is included in file timecons.in. If a user defines this information by the Command Procedure Table it will automatically be included into this file.

To express the power of a resource and to allow for a comparison with other similar resources a resource cycle is expressed by basic cycles given by file cycles.in. If a resource cycle of resource CPU1 amounts

to 1000 basic cycles, and for resource CPU2 it amounts to 10 basic cycles, this means that CPU2 is 100 times faster than CPU1.

In simulation mode, for each resource a queue is established to schedule the processes and devices which access it. The queue is priority-driven and pre-emptive. When consuming a resource a process or device is blocked in the standard mode. In non-blocking mode it remains executable but suspends the action which consumes the resource. After consumption of the resource it receives a message from the queue manager that processing of the action can be continued.

### 3.2.6 Data Exchange Format

For message exchange two types of formats are introduced (see also chapter 2):

- a "short" format which only carries a few data or a command only
- an "extended" format which includes additional ("attached") data in a user-defined format.

The data packet built according to these formats is called a "message".

The "short format" includes the following information:

- information about the sender
- information about the receiver
- information about what the receiver shall do on reception of the message ("command")
- priority of the message
- short information contents (some data)
- format type (short, extended, format of attached data/information)
- length of the complete message (including the attached data)
- timing information
  e.g. when the message was sent initially or actually (in case it passes several modules).

The extended information is added after the end of a short message.

The code generated by the toolset will automatically decode the messages and provide the attached data by a separate byte-array to the user-defined functions. Similarly, code is generated to attach user data automatically.

### 3.2.7 Interfaces

Due to standardisation of input and output interfaces a system's architecture can be built from generic code units. This allows the mapping of logical onto physical channels and to cover redundancy, fault tolerance, reconfiguration and access of heterogeneous physical channels via a unique logical interface. Also, transparent distribution of processes across a network becomes possible.

Fig. 3-3 shows the principal organisation of input and output management.

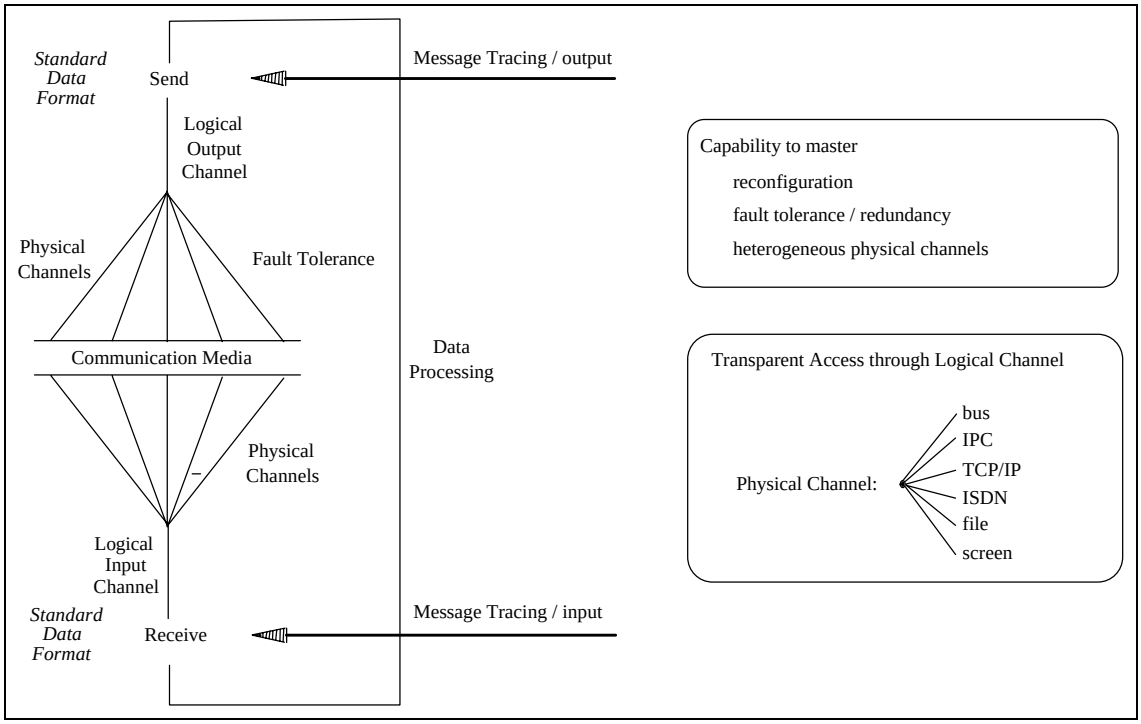


*Fig. 3-3: Standardisation of Interfaces and Management of Input and Output*

## 3.3 How to Establish a New Project

To start a new project a user customises the three principal files

- cmdproc.in

  the "command procedure table"
  a file which includes the formal definition of the operations in terms of behaviour, performance, architecture and topology,

- easysystem.def
  a file which includes configuration and code generation options and possibly explicit declarations of system elements,

- easyconf.h
  a file which includes configuration code generation options to be provided to the C compiler.

Having provided all the engineering and configuration information a user starts the script "createapplfiles" with the project's name as parameter. This script evaluates all the information and builds the program and the environment for compilation, linking, execution and result analysis. It takes

SDL (if desired) and C templates, template of input data files and generates all the required source and data files for the given system configuration.

When the script has executed such steps the system is already executable by the simulator or on a target system and a first feedback can be obtained. Depending on the complexity of the system it takes about 10 to 20 minutes to generate everything from scratch what is needed for execution. By execution graphical figures of data flow (MSC) (see chapter 7), propagation of data over time (timing diagram) (see chapter 7 and reports (coverage, exceptions, injected errors, see chapter 7) are provided. On activation of an option the generator inserts automatically actions for fault injection.

Then, the user can incrementally add functionality in each of the processes by adding and expanding branches according to the commands to be processed After each (structural) extension or modification of the command procedure table or by modification of user-defined functions the system may be subject of validation either by the simulator or on the target system.

Validation support may be extended by interfacing with other tools like ObjectGEODE.. EaSySim II provides library routines by which the source code can be instrumented for resource consumption and performance simulation, generation of Message Sequence Charts (MSC's) and timing diagrams, and a utility which removes the instrumentation automatically. The ObjectGEODE simulator and verifier allow to formally check the message exchange and the behaviour of the system. Interfaces to other formal verifiers can be made available on request.

By the commands which are included in each message the communication between each module is formalised in the sense that

- a set of commands can be defined which are to be processed by a module and which are legal input

  this also implicitly defines what is not a legal input
- a set of output commands (if any) can be correlated with each possible input command forming a command consequence.

Hence, a system's activities are described by the data flow between the modules and the associated commands. This allows to perform automated testing based on the input-output mapping of commands and validation of the data and command flow already in an early development phase when the functionality is not fully implemented, but the principal functionality and the resources are already known.

### 3.4 Formal Specification of Behaviour and Performance

The interaction between a system's elements is defined via the "Command Procedure Table" (CPT) to be provided by file cmdproc.in. The principal organisation of this table is shown by Fig. 3-2. Each line of this table relates the triple (device/process, inital state, incoming command) with a number of actions. An action consists of processing of a user-defined function, issueing of an outgoing command (if any) to a destination process or device (the outgoing command may be sent to the processing device itself), consumption of a resource, and entering of a final state. A number of actions related to (device/process, inital state, incoming command) may be grouped (Fig. 3-4). On reception of a command one group is executed depending on conditions defined by the user (via a macro file) or randomly or sequentially in case of simulation or still missing user code.

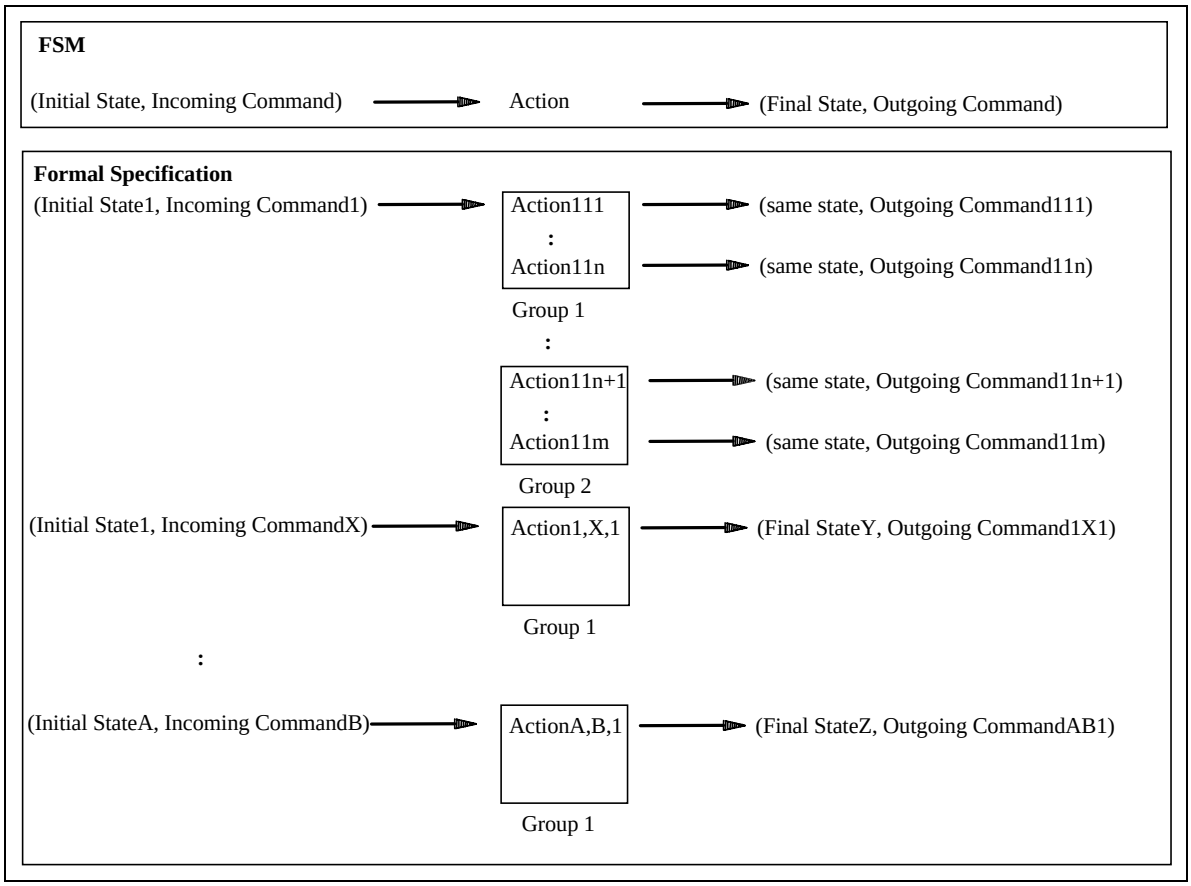


*Fig. 3-4: Detailed Organisation of the "Command Procedure Table"*

The elements of each line are:

- "@" to indicate it is a valid line
- A name of a user-defined function to be executed if (device/process, inital state, incoming command) is matched, the term "none" may be used if no user function shall be called.
- An indicator "u", "i", "I" or "s" telling the system whether a stub shall be generated for this function ("s", "i", "I"), or it is user-provided ("u").

  In case of "s" a dummy body is generated which includes instrumentation for testing and load generation. In caseof "i" the function body includes '#include "funcbody_<device>" and activates the relevant code of a function by compiler switches so that external user code can directly and automatically be included.

  The difference between ("i", "I") and "u" is that in case of "u" the user has to provide the function completely, while in case of "i" or "I" the function prototype, data declarations, copying of input and output data into/from memory and the return are provided by the system. This simplifies the handling for the user.
- Device / process ("src") to which the following information is related

  In case of separate address spaces per process only the relevant lines are taken.[3]

[3] By the configuration option "MEMORYTABLES" and "MEMRESFIELS" of file easyconf.h a user may control whether the data are read in from file or source code for initialisation is generated.

- The number of instances of this device.
  Only the first line per process is evaluated.
- The state (initial state, "initState") in which the command is received.
  "anystate" and asyncstate" may be used here if required.
- The incoming command ("cmdIn")
  The keyword "injerror" may be used as "cmdIn" which means an error is injected for an action related to a group of "cmdIn".
- The outgoing command "cmdOut"

  In case no message shall be issued "noCmd" may be specified.
  The keywords "period", "resetperiod", "timeout", "resettimer" and "deadline" may occur here to initiate specific actions for timers.
- The group indicator (see Fig. 4-2).

  The terms "f", "s", "l" or "o" may appear here.
  "f" indicates it is the first line of a group of several lines, "s" it is the inner line of a group, and "l" it is the last line of a group. "o" is used if the group has only one line.
- The range (minimum, maximum) for repetition of a group in case the user conditions shall not be applied or are still not available. The actual value will be randomly calculated at run-time when the first line of the group is executed. When the upper limit is reached the next group will be executed.

  In case cmdIn="injerror" these two columns specify the probabilty for fault injection: the first column (minimum) the mantissa, the second column (maximum) the exponent yielding a probability of $(col1)*10^{col2}$.
- The priority to process the outgoing command at the receiving process or to transfer the command

  The priority is assigned to the outgoing command. It does not impact the processing of the incoming command. The priority by which an incoming command is processed is taken from the arriving message.

  0 indicates the highest priority, 255 the lowest priority.
- The broadcasting mode ("b","B","s","S").

  By the broadcasting mode the distribution of the comands is controlled. In case of "b" or "B" the command is distributed to all receivers connected to a physical channel, in case of "s" or "S" the command is only distributed to the process/device given by the destination field.

  In case of capital letters the current time is moved into the starttime-field of a message which allows to measure response times.
- The logical channel through which the outgoing command shall be transferred.
  For local processing of a command the reserved, pre-defined term "localCh",
  for handling of the timer actions the term "timerCh" is inserted automatically by the toolset overriding a user-defined value.
- The resource(s) which the device src consumes when executing the action.
  The minimum, mean and maximum values have to be provided by the three following columns.
  In case there is only one instance of a process/device or all instances execute on the same resource this column just takes the one resource.
  In case instances do not share the same resource a list specifies the resource usage. It has the following syntax:

<instance>=<resource {/<instance>=<resource}$^{0..n}$

All instances must occur in this list if they do not share the same resource.

- The minimum, mean and maximum values for time consumption or for timer activities.
  This triple is inserted into file timecons.in and gets the name "autocons<xxx>" where xxx is a unique, numerical identifier per process. The actual value of time consumption is randomly calculated from this range so that the mean value is met.
  If cmdOut is not a reserved command for timers (timeout, rewettimer, period, resetperiod, deadline) the figures express the number of consumed resource (e.g. CPU) cycles. If cmdOut is one of the reserved timer commands, the figures express a duration in seconds.
  At run-time a random value is derived from the given range in case of timeout, period and deadline.
- The destination process / device <dest>.
- The instance number "instd"of the destination process to which the message shall be sent.

  If the value is "0" than the current instance number of the receiving process is taken, if it is >0 the message is sent to (<dest>,"instd") which implies that all instances of <src> will sent the message to the same instance of <dest>.
- The final state <finState>.

  If <dest> = <src> and the actual line is the last line of a group ("o" or "l") then <finState> is entered after processing of this line. Otherwise, for <dest> != <src>, <finState> indicates that is expected that <dest> will receive the outgoing command "cmdOut" in state <finState>.

  The tool will check if (<dest>,<finState>,<cmdOut>) is defined.
- The minimum, mean and maximum value of the attached data.

  During simulation or missing user-defined functions the amount of data to be attached is calculated randomly at run-time from within the given range so that the mean value is met. This is to generate representative data traffic.
- The type of attached data: binary ("BIN") or ASCII ("ASCII").

  This allows to handle and display the attached data in an appropriate manner.

A dash ("-") may be inserted instead of a valid term to get the contents of the corresponding value of the same column of the previous line.

### 3.5 Semantics

The following semantics applies:

#### 3.5.1 Incoming Commands

The term "injerror" may be used in the "cmdIn" field. Then the outgoing command is issued as specified. Following rules apply:

1. Multiple "injerr" commands may occur to the same cmdIn
2. If cmdOut="anyCmd" a command is randomly selected out of the user-defined commands.
3. The rules for <dest> and <finState> remain valid.
4. The two columns specifying the repetition of group execution specifiy the probability for error injection regarding the actual line (see previous section).

### 3.5.2 Outgoing Commands

The terms "noCmd" or "NOP" may be used to indicate that no command shall be issued.

The following terms are related to timing:

1. timeout
   - defines a timeout between the last cmdOut and the cmdIn as given by the timeout line
   - when the associated cmdOut is issued a timer TO<cmdIn> will be started
   - the timeout will raise an exception and generate a command EXCTO<cmdIn>
   - in case of an exception the command EXCTO<cmdIn> will be issued via the given channel (logCh) to the destination device <dest>. The destination device must be prepared to receive EXCTO<cmdIn>
   - execution of the (next) cmdIn for (src,insts) in the specified state <finState> will reset all associated timers
   - the timeout value is taken from the time consumption columns at run-time
   - the unitof these columns is a "second"
   - The rules for <dest> and <finState> remain valid.
   - no resource is consumed
2. resettimer
   - resets the timer TO<cmdIn> and prevents that it will expire.
   - This command is automatically inserted at the beginning of the (initState,cmdIn) sequence if the related option is set at code generation time
   - no resource is consumed
3. deadline
   - defines a deadline w.r.t. to start of the associated cycle of a group
   - the deadline is checked and an error message is created if the deadline is exceeded
   - the deadline refers to the reception of the cmdIn and execution of the first action of the same group
   - the deadline value is taken from the time consumption columns at run-time
   - the unit of these columns is a "second"
   - The rules for <dest> and <finState> remain valid.
   - no resource is consumed when this command is executed
4. period
   - defines a cyclic activity by creating a timer CYCLIC<cmdIn>
   - when the timer CYCLIC<cmdIn> expires a command CYCLE<cmdIn> is issued
   - all commands related to CYCLE<cmdIn> are scheduled cyclically according to the usual rules
   - the associated cmdIn must not appear as cmdOut (tbc)
   - the period value is taken from the time consumption columns at run-time

- the unit of these columns is "second"
- no resource is consumed

5. resetperiod

   - resets the timer CYCLIC<cmdIn> and terminates the cyclic activity.
   - logCh, dest, and instd are ignored
   - no resource is consumed

### 3.5.3 Mapping of Logical Channels onto Physical Channels

This mapping can be done by means of mapping rules provided by easysystem.def.

Several rules may be given. If a rule matches and there is an intersection with following rules, the next rules do not apply any more. If no rule has matched then for mapping between logical and physical channels the default physical channel is associated with the logical channel.

The mapping rules apply to all instances of a device/process.

**Definition of the default physical channel:**

DEFAULTPHYSCHANNEL <coverage> <physical channel>

For "coverage" see chapter 3.4 above.

"physical channel" must be an existing physical channel.

**More sophisticated mapping rules:**

The mapping rules for devices/processes and logical channels may be given in the following manner:

@ch <process/device pattern> <pattern for logical channel(s)> <coverage> <physical channel>

**Example 1:**

```
@ch *   ch2     allroutes       screen1
@ch *   *       allroutes       udp1
```

Above two lines mean:

Line 1: For all devices/processes the logical channel "ch2" is mapped onto "screen1" with coverage "allroutes" (i.e. no constraint on bundeling of channels for fault-tolerance)

Line 2: The remaining logical channels of all devices/processes will be mapped onto physical channel "udp1"

**Example 2:**

```
@ch *   chFT    onceonlyreset   lan1
@ch *   chFT    onceonly        lan2
```

Above two lines mean:

Line 1: For all devices/processes the logical channel "chFT" ("FT=fault-tolerance") is mapped onto "lan1" with coverage "onceonlyreset" (i.e. the coverage is reset and it is indicated that channels are grouped)

Line 2: The same logical channel is also mapped onto physical channel "lan2" with the constraint that it is used only if the previous physical channel is not available

Both rules apply to all processes/devices.

### 3.5.4 Checks

The toolset checks by utility "checktbl" the correctness and completeness of the provided information.

Structural errors in the table lead to an abort of system generation. Other errors or warnings (e.g. incomplete specification, missing devices, states, commands) do not cancel the generation process, but may cause an (recoverable) error at run-time.

Following checks are executed:

- if the triple (source device, initState, cmdIn) is issued as (destination device, finState, cmdOut)

  if not, a warning "code unreachable" is issued.
- if the triple (destination device, finState, cmdOut) is expected as (source device, initState, cmdIn)

  if not, a warning "code is misssing" is issued.
- if exactly one resource is assigned to an instance of a device/process.
- if exactly one exception handler in the desired state is defined for a timeout
- if a cyclic activity is defined as an input in the desired state
- if the grouping is done correctly

  i.e. if the triple (source device, initState, cmdIn) holds for one group defined by ("f", {"s",} "l"} or "o".

  Errors related to this check lead to an abort of system generation.

### 3.5.5 Extensions of the Concept

In addition to the terms introduced before three more terms are allowed to be inserted in the fields of a command line to make the engineer's life easier. However, the drawback of the usage of such terms is that formal checks can only be applied at run-time and not before compilation. Therefore warning messages are issued in case these terms are found.

Such terms are:

**commands:**

udc "user-defined command"

This term is a placeholder to indicate that the actual command is provided by the user at run-time out of the set of allowed commands. This allows to dynamically create outputs.

"udc" is only allowed as outgoing command.

**processes/devices:**

rts "return-to-sender"

This term is used to derive the destination of an outgoing command from an incoming command.

When a command line shall respond to inputs from several devices this generic term allows to send data back to the sender. This may be used to acknowledge reception of data received from a number of devices.

"rts" is only allowed as destination device.

**states:**

anyuserstate

"Anyuserstate" allows to leave the state at the destination open. This option should rarely be used as it turns the approach from formal to informal. However, in some cases it may be justified to apply it.

"anyuserstate" is only allowed as state of a destination and the destination must not be identical with the receiving device.

# 4. The Verification and Validation Strategy

## 4.1 Rational

The verification and validation (V&V) strategy is based on the approach of early system validation (EaSyVaDe, [RD2, RD3b]) and experience with this approach by several pilot applications and real projects. The goal is

- to provide means for risk reduction early in a project's lifecycle,
- to provide support for behavioural and performance validation
- to support V&V over the full lifecycle on host and target
- to provide V&V means which do not necessarily require deep knowledge and understanding of the application
- to reduce the costs for system development.

The HRDMS project [RD02] showed the need for performance validation and the related benefits. The OMBSIM project [RD3b] introduced a formal description of behaviour by means of SDL and combined behavioural and performance validation.

A major step was the introduction of Finite State Machines (FSM's) by SDL and the related verification techniques like simulation, especially exhaustive simulation. However, it turned out rather soon that the powerful verification means fail for practical cases due to state explosion, at least when trying to verify the complete system. This limits the verification to some specific scenarios by means of filtering and constraining the number of system components. It also was recognised that there are slight or major differences between the system as used for verification by simulation and the later target system.

On the other side, it was identified that introduction of performance aspects reduced the number of system states and helped to succeed with exhaustive simulation. Also, it was needed to move such parts of a system to C in order to hide to the tool what was not needed for verification of behaviour and to simplify such parts which remained visible for SDL.

This required to enhance the commercial SDL environment by a lot of features, raising finally the question if with the same effort it would be possible to enhance a C development environment such that it can take benefit of the formal solution as supported by SDL.

Obviously, the strong points of SDL are the use of FSM's and the formal description of behaviour, the related capabilities for verification and the visualisation of the data flow by Message Sequence Charts (MSC). However, this also could be realised in C, because SDL is translated into C for execution. Hence, the idea of interfacing with SDL verification capabilites from C was borne.

Moreover, it was observed during a larger project (CADIS, [RD08]) which was based on SDL and C that still a significant effort was needed for implementation of a system. The high productivity figures observed during the first pilot applications decreased when the functionality of the system increased. The high saving of costs observed at the beginning was related to the automated generation of the process interfaces and communication software, but the expansion of the FSM's still needed to be done manually which decreased the productivity figures significantly.

This lead to the conclusion that a higher degree of automation is needed: the idea of ISG was borne. During the CADIS project a first attempt was made to standardise the construction of FSM'S. Manually, this approach was tested and improved so that finally clear rules existed which allow for automated construction of a system's software.

On the other side by the OPAL project [RD07] experience was gained on automated test stimulation of a system based on SDL. Now, only a final step was needed to combine both features to a unified development and V&V approach.

The MSL project was interested in this approach and this opened the chance to apply the approach continuously from the first phase to the final phase and to gain experience by a real on-board project.

In the first approach the system was intended to be based on SDl, but after some months of discussion the project decided for a pure C environment. Some of the reasons were:

- the need for UDP and an option for a transparent switch between UDP and message queues,
- the high number of processes which were expected and which would have lead to state explosion rather soon and which would have prevented a verfication of the complete system.

This decision initiated a bi-valent strategy: to support the SDL environment and the pure C environment so that a user can select the environment he thinks it is the best one in his case.

### 4.2 The Cornerstones

The cornerstones of the V&V strategy are:

- formal description of behaviour by means of FSM's
- formal description of distribution and resource allocation
- formal description of performance
  (resource consumption, amount of exchanged data)
- formal description of the communication channels

Based on this information the software system can be constructed automatically such that it is immediately excutable. This allows for an incremental development approach. The user specifies on high level the properties of a system (behaviour, performance, distribution, topology) and by ISG an executable is automatically generated. A user may define interfaces to functions for which stubs are generated and into which a user later can plug-in the real functionality.

All this information (currently) has to be provided by a table which may be derived from a spreadsheet. The contents of this table is subject of formal checks due to formal relations defined between the entitities of this table.

Firstly, from this table code can be generated automatically. Secondly, test cases can be derived automatically. Thirdly, error injection can be formalised and automated. Last but not least it is possible to add automatically some functionality about which a user does not need to care himself.

Such automated extensions address e.g. the reset of timeout monitoring or the construction of a command dispatcher.

### 4.3 The Practice

As it is shown by Fig. 4-1 the whole approach is based on the ISG concept implying a formal definition of a system and allowing for automated construction of the software and the V&V environment. This yields two parts: the real system as needed to perform the system operations, and the stimulation and the tracing part.

While the operational part heavily depends (obviously) on the actual application, the (generic) stimulation part remains fixed and only receives information by data tables for details of test stimulation.

Now, for V&V there are the following main aspects to be considered:

- firstly, by test stimulation and tracing of results of the operational part a number of figures can be derived which allow identification of errors (cross-checks on what was defined and what actually is supported)
- secondly, by built-in checks introduced into the operational part compliance with desired behaviour can be checked and inconsistnecies can be detected
- thirdly, the checking, coverage, analysis and reporting concept which is based on the formal definition allows to identify inconsistencies which cannot be deteceted by the lower level represented by "system stimulation" and "system operations".

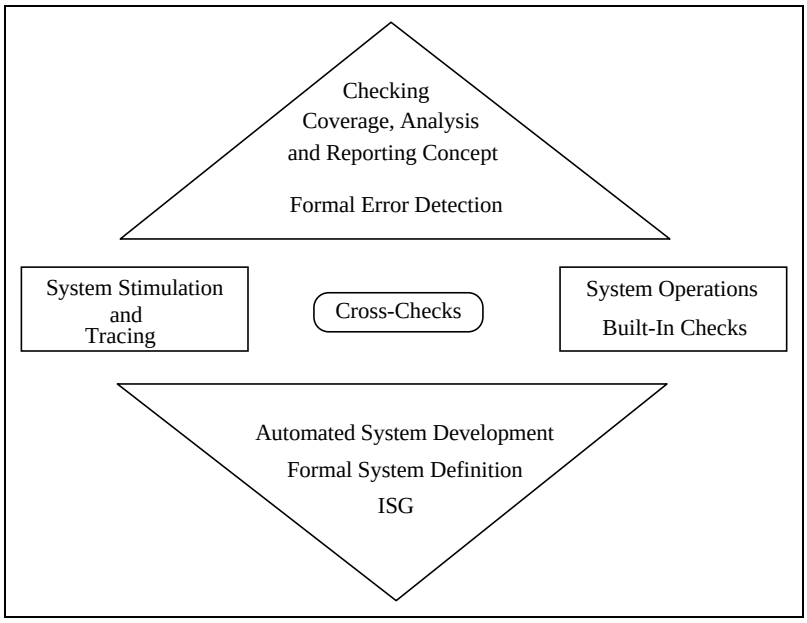


*Fig. 4-1: The Components and Principals of the V&V Strategy*

This "roof" is based on formal relations and allows to identifiy errors even if an engineer and the V&V environment do not have any knowledge of the desired system operations. Hence, the concept enables the V&V environment to identify problems without being asked for.

The identification of errors is symmetric and applies to the infrastructure, the test stimulation and the system operations part. As was mentionned before the environment was derived from existing software, but included a number of new features. So a number of errors were also detected in the V&V part, in fact more than in the application (represented by the command procedure table) due to the large number of options and branches in the V&V and ISG software. And it took a lot of more time to identify such errors in the supporting environment. However, this also indicates the future cost saving because a well-tested infrastructure is available now.

Errors which cannot be identified by the lower level are identified by the checking, coverage analysis and reporting concept. Such cases occur when the system remains quiet like for a response for which no timeout condition is defined or when the wrong instance is addressed and responds. In such cases the coverage analysis identifies a non-covered command line for an instance.

During the first phase of V&V most of the errors will be detected by the lower level or by static analysis of the command procedure table. During this phase the goal is to get the system running at all.

Then during the second phase the goal moves from "getting a reaction from the system" to a higher demand of "getting a complete coverage" of system activities.

### 4.4 More Support by Formal V&V

During the activities for the MSL project the responsible engineer identified that more advantage can be gained by the formal system definition. He recognised that it is very easy to correlate external commands with internal command sequences. This was discussed and the result was that the ISG environment was extended by a feature for automated construction of a command dispatcher which converts external commands into a number of (timed-out) sequences of internal commands.

The benefit of this approach is that also the external commands are now formally defined and conflicting information can be identified. But the higher advantage is that there is now an a-priori / built-in consistency between the external commands and the internal command sequences.

### 4.5 Portability

The software generated by ISG and V&V is completely portable: the user just requests generation of executable code for the desired platforms like UNIX(Solaris/Linux) or VxWorks and starts it on this platform.

This allows to perform the V&V activities in a comfortable and powerful UNIX environment in which the errors can be removed and the system can be verified and validated. Then in a second step the V&V activities can be continued in a real-time envrioment like VxWorks where only errors related to specific real-time constraints should occur, if any.

## 5. The Inputs Required for the V&V Activities

For the ISG approach three basic files are required from which the (software) system is built accordingly and stimulation of the system and its components is derived from.

Such files are:

- the "Command Procedure Table" with (default) filename "cmdproc.in".

  This table describes by its lines the behaviour of the system (Finite State Machines, FSM), the communication between its components, the communication mode and the (estimated) amount of data to be exchanged, its resource consumption (how much of which resource), and its distribution across a network (if decentralised).

  Each action of the system (triggered by an incoming "command") is represented by one line or several lines grouped together. In response to an incoming command a command may be issued (or not), an application function may be executed or an internal service may be asked for.

  As this table defines what the system shall do in repsone to incoming commands (received in a certain state) it is called the "Command Procedure Table" (CPT). This table includes all the information necessary to build an executable (distributed) system and to analyse the feedback from the specified behaviour and performance constraints.

  Also, the correlation of external commands with internal command sequences may be defined. In this case the command handler and dispatcher and test stimulator are derived directly from the information included in this table.

  Moreover, a number of checks can be applied to the table contents because there exist a number of formal relations between the various entities of information which allow cross-checks on consistency and completeness.

- the "System Definition" file with (default) filename "easysystem.def".

  This file includes options which impact additional properties of the system like options for error injection or test stimulation, definition of the network topology and its properties, definition of the starting command and process and so on.

- the "System Configuration" file with (default) filename "easyconf.h".

  Like the System Definition File this file also includes options which impact how the final system looks like. However, such options are related to the implementation language C and hence this file is directly included into the C code. Such options are e.g. switches impacting test printouts, CPU load generation, calibration constants, activation of message queues or UDP and so on.

The only input received from the MSL project was a "big" "Command Procedure Table" defining all of the properties and attributes of the MSL (software) system. A preliminary smaller table was already received after about one month since project's start in order to check the feasibility, the completeness and the common understanding of the approach.

Then a second and nearly complete version was provided (the "big" table consisting of about 500 command lines) which was subject of the V&V activities. This table was updated according to the obtained results of V&V until

- a 100% coverage of command lines and of states was achieved,
- the observed behaviour of the system was as expected,

- no error messages were printed prior to execution or during execution by the toolset and the executing environment.

The MSL project established the command procedure table as EXCEL spreadsheet and exported its contents to ASCII text which was taken as input for system generation and V&V.

From the MSL command procedure table the executable system with all its parameter-files and the packages for the two nodes (CPU'S) were automatically established from scratch within about 40 minutes (including compilation). After about 15 to 20 minutes execution time a coverage of at least 2 per external command was achieved at a rate of 1 external command/s and the the run was automatically terminated. Hence, after about 1 hour a complete evaluation report was available.

After identification of the reported anomalies like flagged errors or "achieved coverage<100%" for command lines and states the table was updated locally (at BSSE) and at the project's site (at KT) in case of small changes, or a new update was sent by e-mail by the project in case of major changes.

A statistic on the size and complexity of the MSL (software) system as represented by this table is shown below.

```
+-----------------------------------------------+
+           ISG Table Statistics                |
+-----------------------------------------------+

Common statistics:

USERPROCESSES=37

GLOBCMDS     =159
SPECCMDS     =66
TOTUSERCMDS  =225
USERSTATES   =41

Statistics for the basic ISG table:

LINES          =476
STATETRANS     =284
OGROUPTRANS    =185
FLGROUPTRANS   =99

EXTCMDLINES    =149
EXTCMDSEQUENCES=149
EXTTOCMDS      =121
TRANSPARENTCMDS=5

Statistics for the expanded ISG table:

ACTIONLINES =534
STATETRANS  =284
OGROUPTRANS =225
FLGROUPTRANS=117
```

The term "basic table" means the table as delivered by the project while "expanded table" means the final table after processing by the ISG toolset which added lines for automated reset of timeouts and for automated injection of commands or for error injection.

There are 37 user-defined processes and an amount of 225 commands by which these processes are communicating. The total number of user-defined states is 41 (enumeration list). 111 states are included in the 37 processes. The number of states of a process ranges from one state up to 36 states in case of the initialisation process.

The basic table includes 476 commanding lines which represent 284 state transitions (including a transition to the same state). There are 185 single-line transitions and 99 multi-line (grouped) transitions.

149 external commands can be received by MSL, of which 121 are correlated with a timeout condition. Amongst such there are 5 transparent commands which are directly forwarded to the destination process while all the other external commands will be pre- and/or post-processed by the command dispatcher.

A small and compressed part (including only the most important information) of the MSL command procedure table is shown on the next page.

For the 149 external commands 447 stubs are automatically generated for pre- and post-processing and syntax checking.

A small part of this command procedure table (part of the iniitalisation procedure) is shown on the next page, an explanation is given on the page after the next page.

| Line | Source Process | # | InitState | Incoming Command | Processing Function | Command To be sent | Channel | CPU Node | TO | Destination Process | # | State at Destination |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | sysinit | 1 | anystate | sub_initcmd | init_digdae1 | sub_initcmd | ch1 | fcu | - | digdae | 1 | samestate |
| 2 | sysinit | 1 | state0 | subinitreturn | none | timeout | ch1 | fcu | 2 | sysinit | 1 | anystate |
| 3 | sysinit | 1 | anystate | sub_initcmd | none | nocmd | ch1 | fcu | - | sysinit | 1 | state0 |
| 4 | sysinit | 1 | state0 | subinitreturn | none | resettimer | localch | fcu | - | sysinit | 1 | samestate |
| 5 | sysinit | 1 | state0 | subinitreturn | init_digdae2 | sub_initcmd | ch1 | fcu | - | digdae | 2 | anyuserstate |
| 6 | sysinit | 1 | state0 | subinitreturn | none | timeout | ch1 | fcu | 2 | sysinit | 1 | anystate |
| 7 | sysinit | 1 | state0 | subinitreturn | none | nocmd | ch1 | fcu | - | sysinit | 1 | state1 |
| 8 | sysinit | 1 | state1 | subinitreturn | none | resettimer | localch | fcu | - | sysinit | 1 | samestate |
| 9 | sysinit | 1 | state1 | subinitreturn | init_anadae1 | sub_initcmd | ch1 | fcu | - | anadae | 1 | anyuserstate |
| 10 | sysinit | 1 | state1 | subinitreturn | none | timeout | ch1 | fcu | 2 | sysinit | 1 | anystate |
| 11 | sysinit | 1 | state1 | subinitreturn | none | nocmd | ch1 | fcu | - | sysinit | 1 | state2 |
| 12 | sysinit | 1 | state2 | subinitreturn | none | resettimer | localch | fcu | - | sysinit | 1 | samestate |

The contents of the columns is:

| | |
|---|---|
| Source process | the process receiving a command |
| # | the number of instances of the source process |
| InitState | the state of the source process in which it receives the incoming command |
| Incoming command | a command from another (or the same process) triggering an action / state transition |
| Processing Function | an aplication function to be called when the command line is executed |
| Command to be sent | a command possibly to be sent at completion of processing of the actual command line |
| Channel | the (logical) channel through which the command shall be sent |
| CPU / Node | the resource/ CPU on which the action is executed |
| TO | the value of the timeout if "command to be sent" is "timeout" |
| Destination process | the process which shall receive the outgoing command |
| # | the instance of the destination process |

State at the destination process the state in which the destiantion process shall receive the outgoing command

The meaning of these lines is:

Line 1 The initialisation process receives the sub_init_cmd and forwards it to the "digital daemon #1"
Line 2 It expects the response "sub_init_return" within a timeout condition of 2 s and initiates a timeout
Line 3 It switches to "state0"

Line 4 The response from the digital daemon is received and the timer is reset
Line 5 It sends another sub_init_cmd to the "digital daemon #2"
Line 6 It expects the response "sub_init_return" within a timeout condition of 2 s and initiates a timeout
Line 7 It switches to "state1"

Line 8 The response from the digital daemon is received and the timer is reset
Line 9 It sends another sub_init_cmd to the "analog daemon #1"
Line 10 It expects the response "sub_init_return" within a timeout condition of 2 s and initiates a timeout
Line 11 It switches to "state2"

Line 12 The response from the analog daemon is received and the timer is reset
and so on ...

# 6. The Performed V&V Activities

## 6.1 Overview

The V&V activities were performed by the following steps:

- **visual checking of the table contents**
- **check of the consistency and completeness** of the table contents by the "checktbl" tool
  (check of compliance with the command table syntax and semantics)
- **execution of the generated (software) system**
  - -- check of text output for error messages
    like "illegal command for the present state"
  - -- check for completeness of (expected) state transitions
    like for the initialisation process
  - -- check of the message sequence charts for error messages and anomalies
    either ASCII file or graphics
  - -- check of the timing diagrams for unexpected patterns
  - -- check of the coverage figures
    coverage of states and command lines
  - -- check of CPU and network utilisation
  - -- check of timer load
  - -- check of comamnd buffer load
  - -- check for exceptions
  - -- check of response times
- **automated stimulation of all processes of the system**
  (except some special ones like sysinit, cmdhandler, digdae and anadae)
  observation of system behaviour under stress testing
- **automated error injection for all processes of the system**
  (except some special ones like sysinit, cmdhandler, digdae and anadae)
  - -- observation of achievable coverage
  - -- check for error messages
  - -- check of occured exceptions

The V&V activiities were performed on a UNIX/Solaris/UltraSparc platform.

Pre-tests on the target SPLC platform, especially performance tests have been executed and integration of a simplified system version was performed.

Full integration of the system on its target platform is expected in about two weeks. Then realistic figures on CPU and network utilisation and on response times will be available. Unfortunately, such tests could not be executed before the official delivery date of this document.

### 6.2 Categorisation of the Identified Errors

The V&V activities were performed in the sequence described in the previous section. The identified errors are classified into five categoreis:

- errors detected by visual checks and tool support at pre-run-time
- errors detected by execution due to error messages and abort of sequences
- errors detected at post-run-time by coverage analysis
- potential problems identified by stress testing
- potential problems identified by error injection.

The type and number of detected errors not only depends on the application and its complexity, it also depends on the development and V&V environment. As can be seen by the report given below some errors are strongly related to the operations by which a system's behaviour is defined: in this case a significant number of errors was related to copy/paste operations when the contents of the copied line was not completely changed. Also, frequently the wrong destination of a message was specified, in most the wrong instance number.

Vice versa, by the absence of some error types it may be concluded that the applied approach prevents such type of errors.There was no error related to a communication channel because ISG is in charge of managing the communication. No false alarm occured due to a missing reset of a timeout because this is handled automatically.

Regarding the high complexity of the MSL software system (37 processes, about 230 commands, 111 states) the number of identified errors seems to be relatively low.

The type and number of errors found at pre-run-time, run-time and post-run-time are an indication of how well the set up of the system definition can really be done: no very serious errors were detected. About 50% of the errors were related to copy-paste of comamnd lines and forgotten changes. A minor part was related to the complexity of the required behaviour, e.g. when more than one process level was involved to provide a feedback, i.e. a process receiving a command has to involve another process before it can respond. Another error type was that it was forgotten to respond (because another activity was initiated) or a wrong destination was specified.

#### 6.2.1 Visual Checks and Automated Checks at Pre-Run-Time

These actvities checked the global structure of the command lines and the correct understanding of the semantics. At a minor part errors were detected by visual checks. Most of the errors were reported by the checking tool.

Typical errors detected by visual checks are:

- repeated execution of the same UDF (user-defined function) within a group of command lines related to the same incoming command
- wrong use of a '-' to take the contents of the same column of the previous command line.
- wrong definition or selection of the destination state in case of timeout requests
- wrong position of the state transition within a group (not performed by last line)

- wrong destination instance (twice "1" instead of "1" and "2")
  if not detected by visual checks such an error will be detected by coverage analysis

To a major part errors were detected at pre-run-time by the checking tool. Such errors are:

- the combination "source process / initState / incoming command" does not occur as "destination process / destination state / outgoing command" (non-reachable code)
- the combination "destination process / destination state / outgoing command" does not occur as "source process / initState / incoming command" (missing code)
- conflicting assignment of resources (CPU's)
- inconsistencies between a specified timeout of an external command and the responding command of the process ("nocmd" instead of an ack-command)
- structural inconsistencies of groups

Also, the C linker detected an error due to multiple use of the same symbol name

- multiple use of the same name of a UDF for different processes (by copying)
  recognised during compilation

Some features which were introduced due to operational needs or for higher user comfort reduced the degree of formality so that the related errors could no longer be detected at pre-run-time but at run-time only. Such features are:

- RTS return to sender

  This feature was needed for operational reasons as the same input command could come from different sources (e.g. sub_init_return in case of sysinit), otherwise a high number of specific cases and lines would be needed.

  In a later version by more sphisticated analysis most of the related errors can be detected at pre-run-time again.

- UDC user-defined command

  This feature allows to keep the triplet "destination process / destination state / outgoing command" completely open and hence does not allow for any pre-run-time checks.

  It is also needed for operational reasons e.g. to cover issueing of an arbitrary number of commands.

- anyuserstate

  This feature was introduced for higher user comfort so that a user may change arbitrarily the state literals in a destination process e.g. in case a new state must be inserted. If a new enumeration sequence is defined he does not need to update the correlated lines.

  Clearly, this decreases the level of formality down to informal level only.

Also, such features impact automated test stimulation because the receiver of a message cannot be identified immediately. However, by more sophisticated analysis this problem can also be solved by a later version.

### 6.2.2 Errors Detected by Execution

Most of the errors were detected by execution either because an error message was issued (in the log-file or in the MSC) or the expected coverage was not achieved. In cases of complex data flow the analysis of the graphical MSC's was very helpful e.g. to understand that and why two ack's had been sent etc..

Following error types and errors were detected by execution of the system and issueing of error messages:

- on reception of the initialisation command the process forwarded it to the next lower level but forgot to respond to the initialisation process, this lead to an abort of the initialisation process
- four processes with one instance used four instances of a daemon

  As the procedures were similar for each of the processes the command lines were copied but the instance number of the destination process (1..4) was not changed accordingly
- For the responding process a fixed instance number (1..4) was defined as destination (instead of 0). This caused to forward the response to the wrong destination and not to the actual sender (instance number 0 forwards the message to the instance which sent the message)
- During the initialisation procedure the wrong instance of two processes was specified, this was also a consequence of copying of a command line, but not completely updating it.
- inconsistent handling of transparent and non-transparent commands by the (generic) command dispatcher[4] regarding timeouts
  this was detected by internal (limit) checks
- for a group of nine command lines "RTS" was used as destination instead of forwarding the incoming message to nine other processes. Hence, the sender (the sysinit process) received nine responses, and the other processes were not initialised.

  Hence, the initialisation process received more ack's than foreseen and it progressed faster through the FSM as expected. The additional ack's caused error messages because they were not expected

### 6.2.3 Errors Detected by Coverage Analysis

Following errors were detected by coverage analysis:

- non-covered command lines
  - -- an error in the mapping of external onto internal commands and instances of processes

    the command line of the instance was not executed (occurred twice)
  - -- missing ack's in case of reception of (external) commands
  - -- the O/G command "set" was used twice (by copying, set/set instead of set/reset)
  - -- the wrong instance number was specified, therefore an instance did not receive the message
- non-covered states
  - -- wrong specification of the instance number

    therefore a state transtion always occured for the same instance instead for each of the four instances
  - -- a process was not initialised (it was not considered by the initialisation procedure)
  - -- periodic activities were not started

    The understanding was that they are automatically started by the system, but a user-defined input is needed to start them

[4] the generic command dispatcher is part of the ISG / V&V environment

-- a wrong response (command) was defined: this command is unknown at the receiving process

-- a process sends erroneously a respone to itself: this command is not known as input

### 6.2.4 Errors and Anomalies Detected during Stress Testing

Such anomalies have been detected, but they may not necessarily be interpreted as errors:

- "state not found"

  A test input should be generated for a user-defined state but no command line was found for this user-defined state (the implicitly, pre-defined default init-state): the process did not implement a FSM, which is allowed

- multiple execution of a command line which iniitated a cyclic activiy

  This lead to an overload situation for several processes. However, it may be justified not to protect against multiple execution of such a line, e.g. because this cannot happen during (normal) system operations.

- sending a command to itself because destination is "RTS" and the command was initiated by the process itself.

  This also lead to an overload due to a never-ending cycle.

### 6.2.5 Observations Made In Case of Error Injection

It was observed (see chapter 7) that the system is sensitive for lost commands. The coverage decreases rapidly at low probabilites for loosing a command. This will be considered by the project.

## 6.3 The V&V Phases

The performed V&V activities can be divided into three principal phases:

- the start-up phase

  This was the phase from handing-over of the command table until successful completion of the initialisation procedure. This procedure consisted of (initially) 35 steps and was updated and changed in its structure a number of times until the post-initialisation state could be reached.

- the coarse pre-operational phase

  During this phase the normal operations of the system were checked e.g. if the cyclic activities were started correctly during initialisation

- the refined pre-opertional phase

  Now, the injection of external commands was activated and their distribution in the system was checked.

  By end of this phase the principal behaviour and communication was believed to be correct.

  Up to this phase mostly MSC's and log-files were used for error detection and identification.

- the operational phase

  All the external commands were injected now and the analysis of coverage came in. A number of new bugs were identified when the the lines and states were checked for which less than 100% coverage were achieved.

- the finalisation phase

  Having succedeed with the principal V&V of behaviour and communication time could be spent to have a look on performance figures like CPU and network utilisation and response times.

- the stress testing and error injection phase

  While the previous phases aimed to prove the correctness (absence of errors) these phases aimed to prove presence of errors.

  A number of inadvertent inputs were identified and a sensitivity regarding loss of messages.

## 6.4 Effort

The effort needed for fixing and removal of bugs was higher at the beginning than at the end. Due to the novalty of the approach a learning phase at the beginning was needed by the user.

### 6.4.1 Start-up

When the final "big" command table was received, by visual inspection a number of errors were immediately detected. Also, the user himself had identified some open points to be discussed and clarified.

The first update of the comamnd line table addressed nearly the whole table and took about 5 hours for a number of iterations for which also the reports from the checking tool were considered until finally all significant error messages and warnings disappeared.

### 6.4.2 Follow-On

Most of the bugs were identified by error messages, data flow and coverage analysis and could be fixed within approximately 1 hour. The update of the table took about 15 minutes in most cases up to about 30 minutes in some cases even when a number of lines were effected.

### 6.4.3 Feedback for Improved Support

When a rather long time was needed for identification of an error the reason was analysed and the ISG support was improved by adding more analysis capabilities, e.g. display of states or command contents in the MSC, printing of non-covered states and command lines etc.

### 6.4.4 Update of Coomand Procedure Table vs. Corrective Maintenance of Infrastructure

The MSL application was the first real and big application for the ISG approach and the related V&V activities. Therefore not only the inputs from MSL were checked, but the underlying ISG infrastrucutre was simultaneously checked. In fact, the MSL application was very challenging for this infrastructure: it applied every feature which was allowed.

It turned out that the identification and removal of bugs in the infrastructure took about 10 to 20 times more time than the identification and removal of bugs from the command procedure table. This ratio gives an impression of what time is saved when an implementation relies on the ISG approach and not on a conventional implementation from scratch.

# 7. Evaluation of the Results

A representative extract of the results is given on the next pages. This information shall help to follow the results of the V&V approach and the drawn conclusions.

The graphical figures 7-1 - 7-7 of section 7.1 shall illustrate which graphical support is available.

The following sections include detailed figures of the evaluation report for the "operational case" and the most important figures from the "stress testing" and "error injection" cases. For each case the relevant reports are discussed. The reports are divided into two principal classes:

- reports on coverage
- reports on performance.

The inputs for the reports and the graphical tools are generated during each operational mode of the system: on host or on target in the desired modes of distribution, either in simulation or real-time mode. This feature allows full visibility on what the actual system status is on the actual platform and provides capability for comparison of figures between host and target configurations.

The "operational case" is a test for which the ethdae (Ethernet daemon) and the mildae (MIL bus daemon) were stimulated with an average rate 1/s and 1/20s respectively of external commands. The test was terminated when a minimum coverage of at least 2 was achieved for all external commands. This required random injection of about 800 to 1600 external commands.

At this coverage of issued commands a corresponding coverage of 100% for command lines and states should be observed. However, for some good reasons which can be justified in some cases a coverage of less than 100% may be sufficient. Such cases are e.g. when a command line is only executable by one instance or when the command line shall handle an exception which did not occur.

The "stress testing" case is a test where all processes are automatically stimulated by command injection where the commands are selected according to their present state. The sysinit, mildae, ethdae and cmdhandler processes were not stimulated this way because of their specific role. To stimulate the sysinit process is not meaningful because it needs to execute a certain step sequence which was already verified. The other three processes are involved in command injection and therefore there is no need to stimulate them. The stress testing test imposed high load on the system and a lot of "nonsense" activities which lead to inadvertent inputs.

The "error injection" case is a test for which all outgoing command are suppressed at a certain probability. The resulting coverage gives an indication on to which degree the system remains in a useful operational state in case information is lost in the system.

Above cases have been selected out of the large variety of possible test modes the ISG environment offers.

## 7.1 The Graphical Presentation Capabilities

Figs. 7-1 - 7-3 show so-called "Timing Diagrams" which visualise the occurence of events/exchanged commands over time for each process. The contents of each command message can be displayed by clicking on an event (Fig. 7-2). Such type of diagrams is helpful to check the propagation of information over time through the different processes. Singular events and periodic patterns can best be identified and checked, also their timeliness. Fig. 7-3.2 shows a pattern characteristic for the initialisation sequence.

Figs. 7-4 - 7-7 show the data flow by means of MSC's (Message Sequence Charts) of which the tracing and visualising capapbilities were extended and improved: more than the usual information is displayed by a format defined by the user. Hence the current state, the number of the executed command line, the

incoming and outgoing command, sender and receiver can be displayed. Also the processing times are displayed: the "starttime" at which a certain command sequence was initiated or the time of last processing ("acttime"). This way response times can be collected. By selecting a certain starttime all events related to this sequence will be displayed, only. This feature is also available for the timing diagrams.

Fig. 7-4 gives by its large compression factor an impression of the communication partners and communication over time. Figs. 7-5 - 7-6 show some snapshots of the overall data flow.

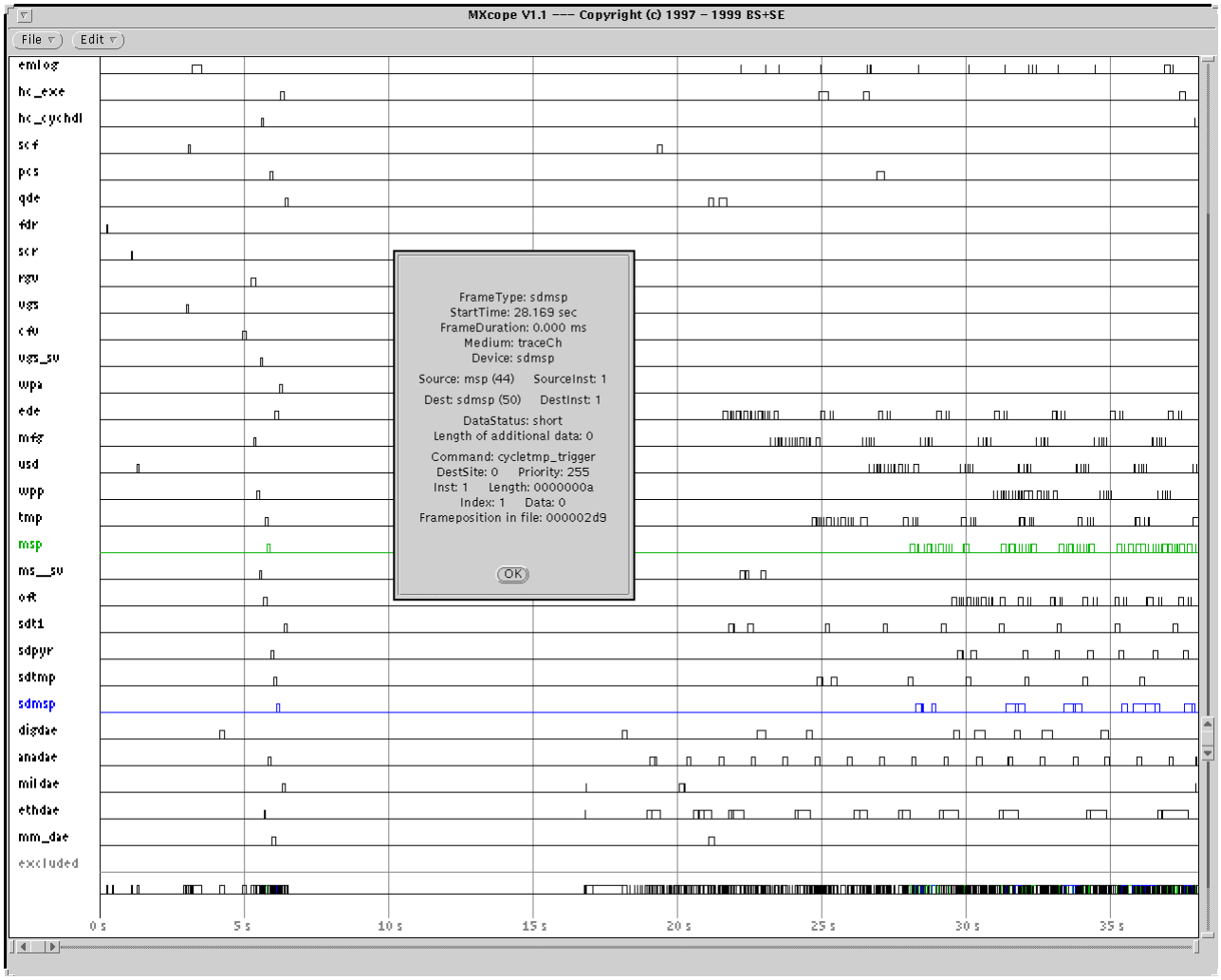


*Fig. 7-1: The Start Sequence of the System / Timing Diagram*

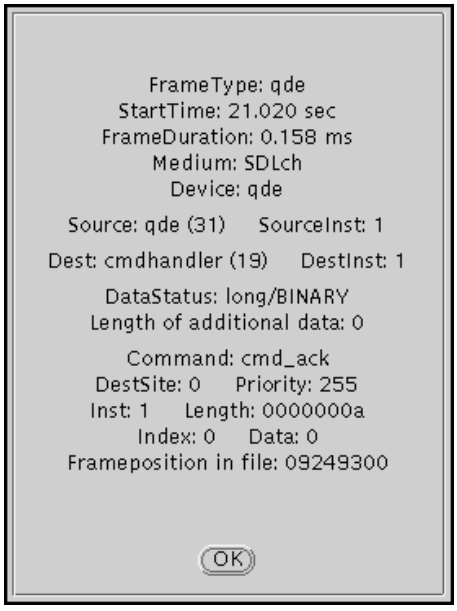


*Fig. 7-2:The Contents of a Message*

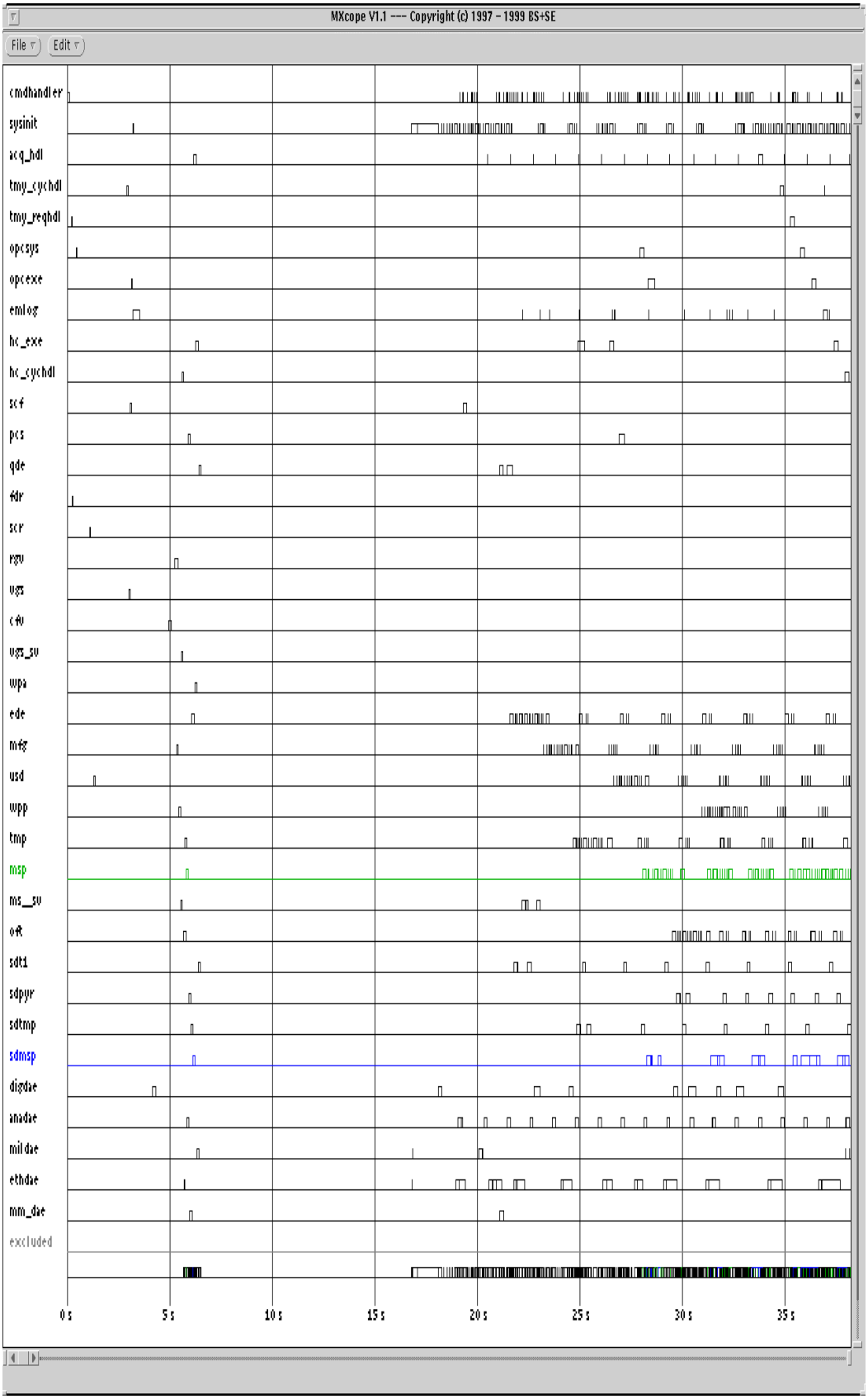


*Fig. 7-3.1: The MSL Initialisation Sequence / The Complete List of Processes*

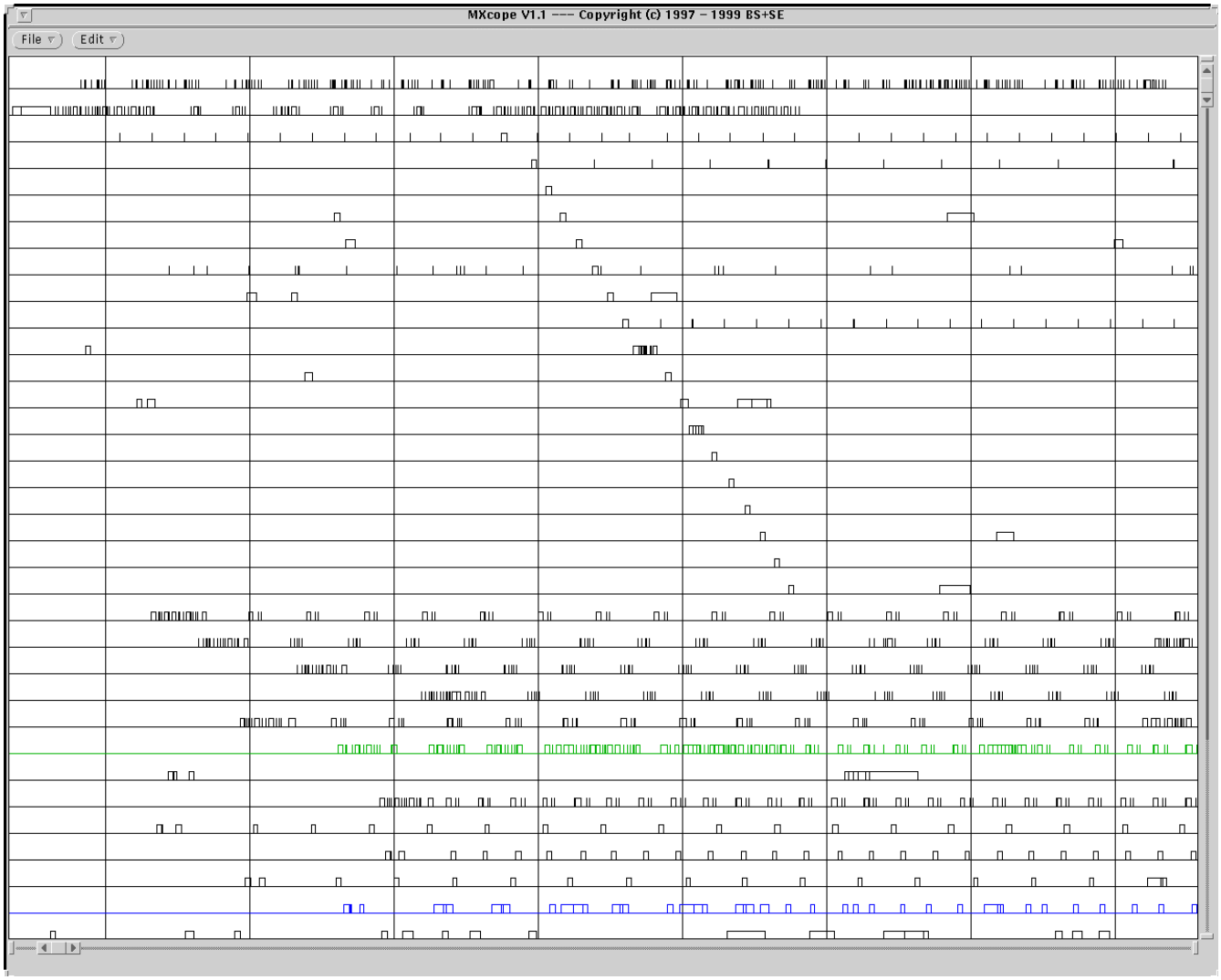


*Fig. 7-3.2: The MSL Initialisation Sequence / Timing Diagram: A Portion of the Initialisation Pattern*

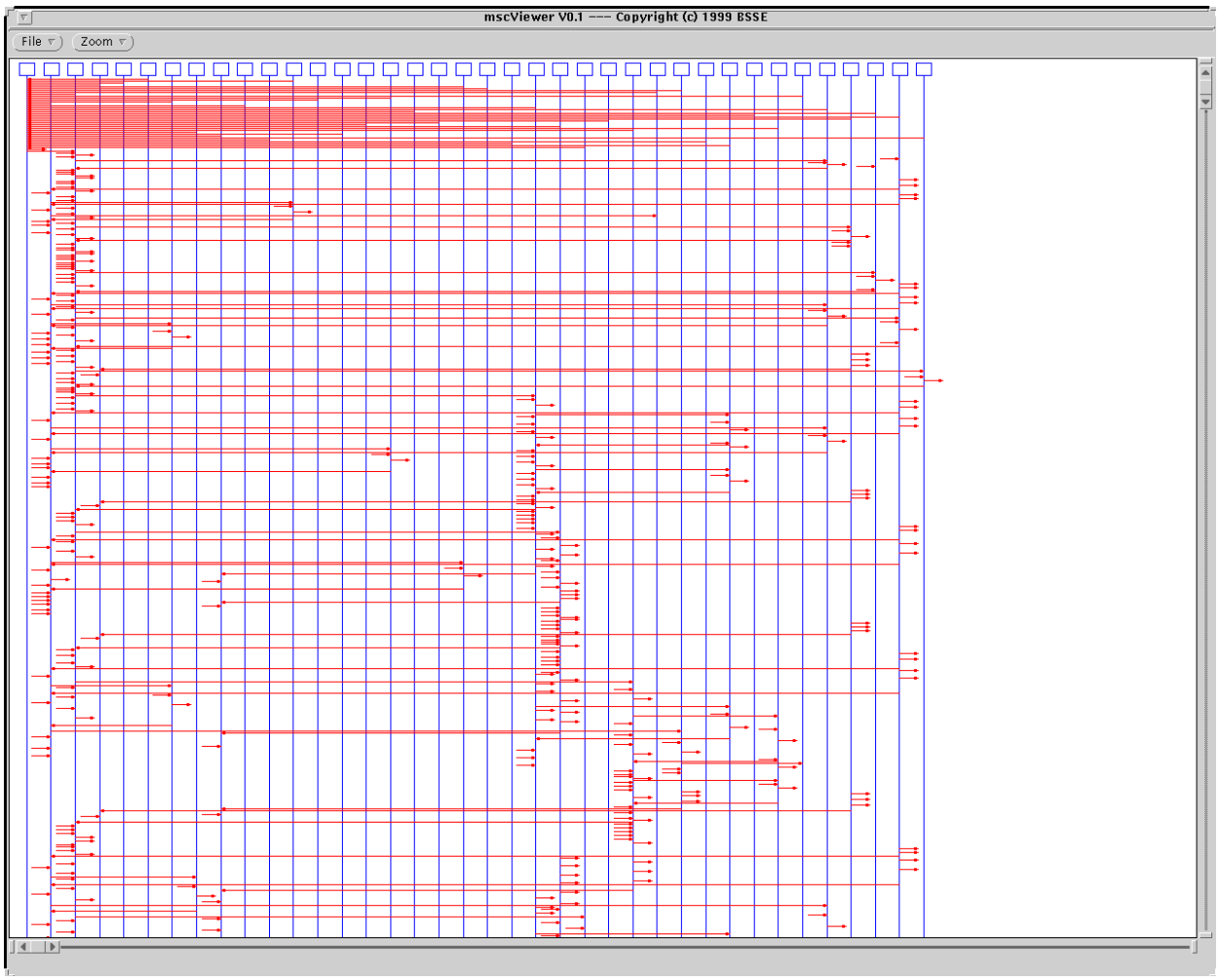


*Fig. 7-4: Overall MSC View on (Initial) System Activities*

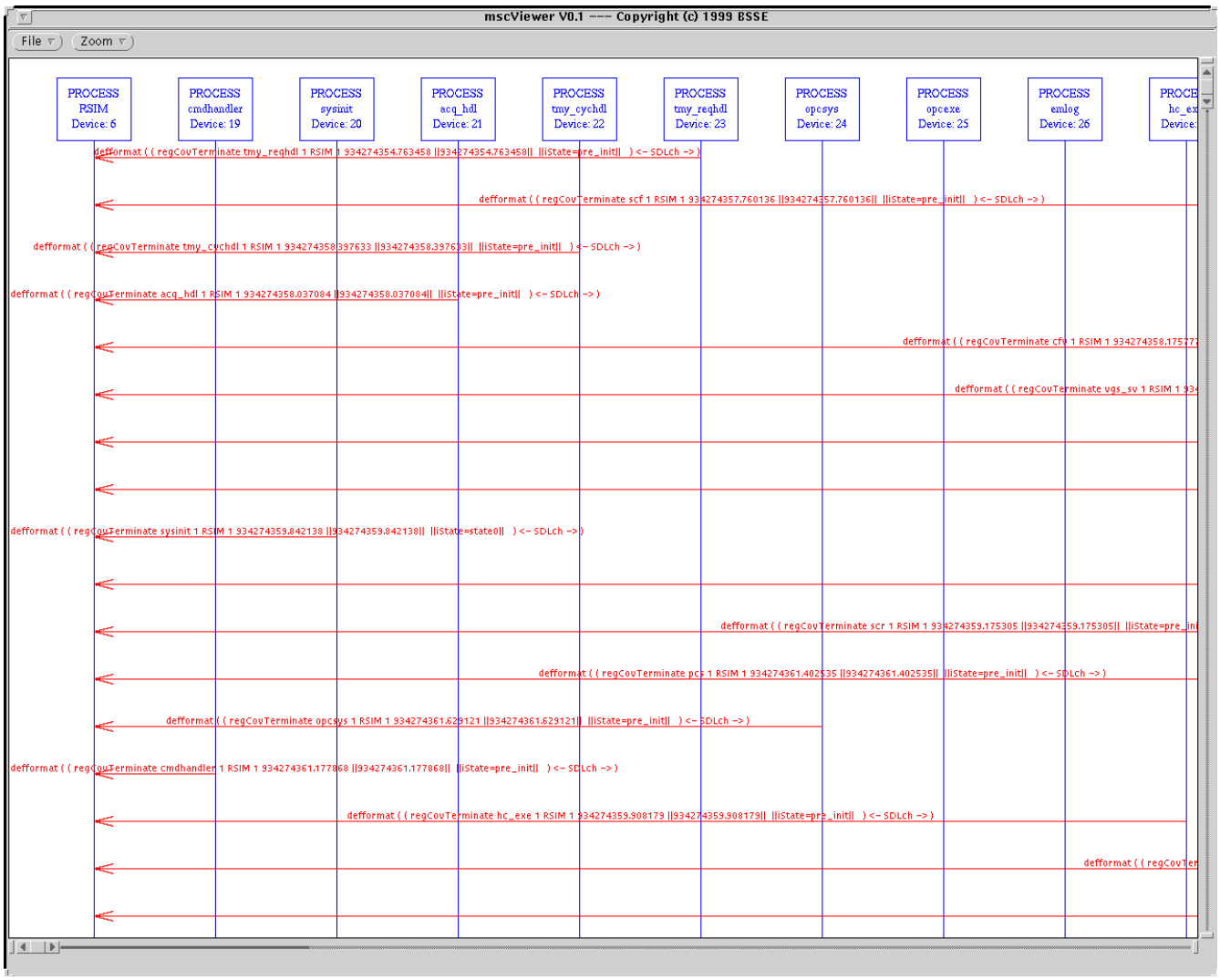


*Fig. 7-5: First Part of System Start Shown by an MSC*

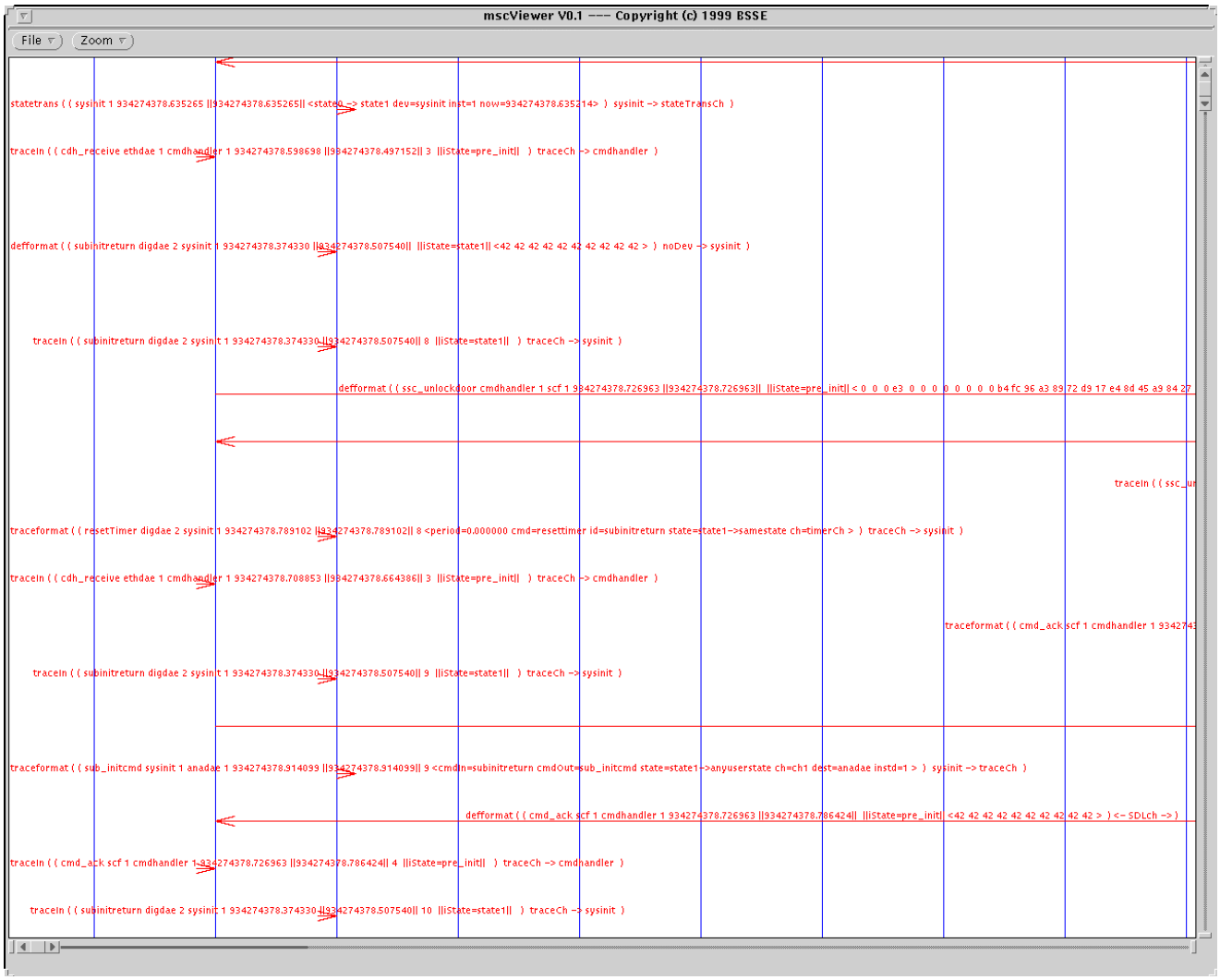


*Fig. 7-6: Sample Operational Sequence (1) / MSC View*

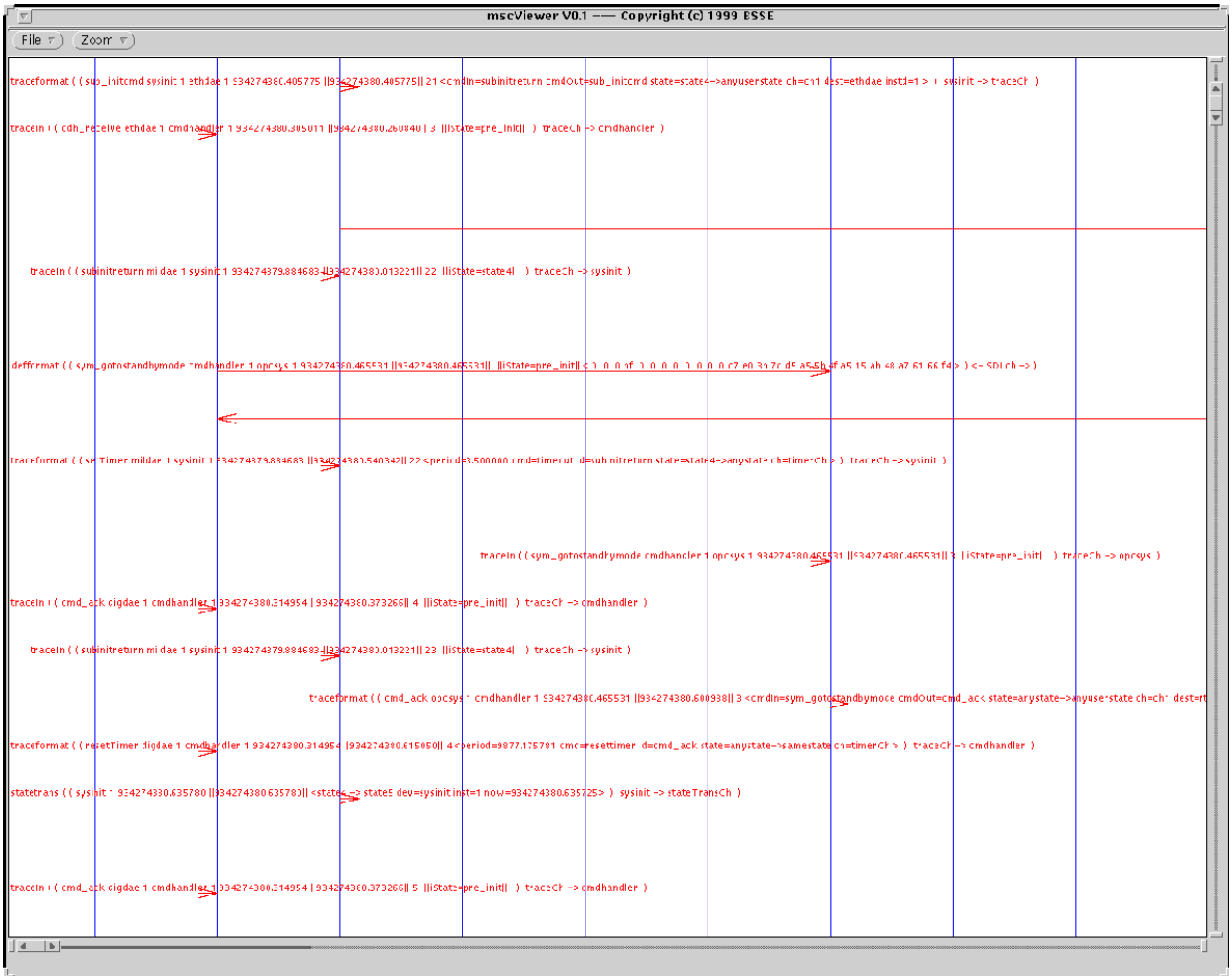


*Fig. 7-7: Sample Operational Sequence (2) / MSC View*

## 7.2 Comparison of Expected and Observed Behaviour

The subset of information of a command line as shown below is more or less equivalent to a line of a MSC. Therefore the simplest way to check for the expected behaviour is to compare each line of the command procedure table with a line of a MSC in graphical or (as shown below) in textual form.

The first block shows the command lines and the second block the corresponding MSC lines.

```
sysinit    1 anystate    sub_initcmd     init_digdae1 sub_initcmd ch1     fcu  -    digdae     1 samestate
sysinit    1 anystate    sub_initcmd     none         nocmd       ch1     fcu  -    sysinit    1 state0

sysinit    1 state0      subinitreturn   none         resettimer  localch fcu  -    sysinit    1 samestate
sysinit    1 state0      subinitreturn   init_digdae2 sub_initcmd ch1     fcu  -    digdae     2 anyuserstate
sysinit    1 state0      subinitreturn   none         timeout     ch1     fcu  2    sysinit    1 anystate
sysinit    1 state0      subinitreturn   none         nocmd       ch1     fcu  -    sysinit    1 state1

sysinit    1 state1      subinitreturn   none         resettimer  localch fcu  -    sysinit    1 samestate
sysinit    1 state1      subinitreturn   init_anadae1 sub_initcmd ch1     fcu  -    anadae     1 anyuserstate
sysinit    1 state1      subinitreturn   none         timeout     ch1     fcu  2    sysinit    1 anystate
sysinit    1 state1      subinitreturn   none         nocmd       ch1     fcu  -    sysinit    1 state2

sysinit    1 state2      subinitreturn   none         resettimer  localch fcu  -    sysinit    1 samestate
```

```
PROCESS sysinit
IN         ( sub_initcmd   RSIM    1 sysinit 1 934480846.074498 934480846.074498  initState=state0 ) FROM RSIM
OUT        ( sub_initcmd   sysinit 1 digdae  1 934480846.189218 934480846.142911  initState=state0 ) TO   digdae

IN         ( subinitreturn digdae  1 sysinit 1 934480846.189218 934480848.218819  initState=state0 ) FROM digdae
ACTION     ( resetTimer    digdae  1 sysinit 1 934480848.439230 934480848.439230
OUT        ( sub_initcmd   sysinit 1 digdae  2 934480848.538979 934480848.499130  initState=state0 ) TO   digdae
STATETRANS (               sysinit 1          934480848.789857 934480848.789857  state0 -> state1

IN         ( subinitreturn digdae  2 sysinit 1 934480848.538979 934480848.645219  initState=state1 ) FROM digdae
STATETRANS (               sysinit 1          934480849.295893 934480849.295893  state1 -> state2
OUT        ( sub_initcmd   sysinit 1 anadae  1 934480849.045533 934480848.989284  initState=state1 ) TO   anadae

IN         ( subinitreturn anadae  1 sysinit 1 934480849.045533 934480849.146193  initState=state2 ) FROM anadae
```

## 7.3 Consistent Correlation of External with Internal Commands

The following blocks show the full contents of a command line. The first block includes command lines which are not correlated with external commands. The second block shows correlations with external commands. To show the complete line a reduced font size is needed Therefore these lines are repeated in the third block at a larger fontsize, but less important information is suppressed.
For a description of the contents of a command line please refer to chapter 3.
The correlation of external ground commands with on-board commands is done by adding three columns to a command line (at the right side): the response timeout in seconds, the mapping of external commands onto internal comamnds and instances and a sequence number which gives the order by which a number of on-board commands shall be executed when the (single) external ground command is received.
Due to the combination with timeouts a commanding sequence related to an external command may be broken down into a number of sequences each terminating with a timeout condition for a response. When the response is received the next part of the whole sequence is processed. If no response is received in time an exception is raised.
This way the command handler can be kept generic and the tables which are needed for command dispatching can automatically be generated from the command procedure table.
This saves a lot of effort and ensures a priori consistency between external and on-board commands.

```
s init_digdae1 sysinit 1 anystate sub_initcmd   sub_initcmd f 1 1  255 B ch1     fcu 100 100 100 digdae  1 samestate    80 100 120 bin
s none         -       - -        -             nocmd       l 1 1  255 b ch1     fcu 100 100 100 sysinit 1 state0       80 100 120 bin
s none         -       - state0   subinitreturn resettimer  f 1 1  255 B localch fcu 100 100 100 -       1 samestate    80 100 120 bin
s init_digdae2 -       - -        -             sub_initcmd s 1 1  255 B ch1     fcu 100 100 100 digdae  2 anyuserstate 80 100 120 bin
s none         -       - -        -             timeout     s 1 1  255 b ch1     fcu 3.5 3.5 3.5 sysinit 1 anystate     80 100 120 bin
s none         -       - -        -             nocmd       l 1 1  255 b ch1     fcu 100 100 100 -       1 state1       80 100 120 bin
s none         -       - state1   subinitreturn resettimer  f 1 1  255 B localch fcu 100 100 100 -       1 samestate    80 100 120 bin
s init_anadae1 -       - -        -             sub_initcmd s 1 1  255 B ch1     fcu 100 100 100 anadae  1 anyuserstate 80 100 120 bin
s none         -       - -        -             timeout     s 1 1  255 b ch1     fcu 3.5 3.5 3.5 sysinit 1 anystate     80 100 120 bin
s none         -       - -        -             nocmd       l 1 1  255 b ch1     fcu 100 100 100 -       1 state2       80 100 120 bin
s none         -       - state2   subinitreturn resettimer  f 1 1  255 B localch fcu 100 100 100 -       1 samestate    80 100 120 bin
```

```
s TMP_RunUp          TMP 2 operational TMP_RunUp        CMD_ack      f 1 1 255 b ch1 - 100 100 100 RTS   1 anystate    80 100 120 BIN 2 1=TP1_RunUp/2=TP2_RunUp            1
s none               -   - -           -              TMPdae_TXcmd s 1 1 255 B ch1 - 100 100 100 sdtmp 0 operational 80 100 120 BIN
s none               -   - -           HK_data        timeout      l 1 1 255 b ch1 - 0.4 0.4 0.4 TMP   0 anystate    80 100 120 BIN
s TMP_RunDown        -   - -           TMP_RunDown    CMD_ack      f 1 1 255 b ch1 - 100 100 100 RTS   1 anystate    80 100 120 BIN 2 1=TP1_RunDown/2=TP2_RunDown        1
s none               -   - -           -              TMPdae_TXcmd s 1 1 255 B ch1 - 100 100 100 sdtmp 0 operational 80 100 120 BIN
s none               -   - -           HK_data        timeout      l 1 1 255 b ch1 - 0.4 0.4 0.4 TMP   0 anystate    80 100 120 BIN
s TMP_SwitchMotor    -   - -           TPM_SwitchMotor CMD_ack     f 1 1 255 b ch1 - 100 100 100 RTS   1 anystate    80 100 120 BIN 2 1=TP1_SwitchMotor/2=TP2_SwitchMotor 1
s none               -   - -            -             TMPdae_TXcmd s 1 1 255 B ch1 - 100 100 100 sdtmp 0 operational 80 100 120 BIN
s none               -   - -            HK_data       timeout      l 1 1 255 b ch1 - 0.4 0.4 0.4 TMP   0 anystate    80 100 120 BIN
```

```
s TMP_RunUp          TMP 2 operational TMP_RunUp        CMD_ack      RTS   1 anystate    BIN 2 1=TP1_RunUp/2=TP2_RunUp            1
s none               -   - -           -              TMPdae_TXcmd sdtmp 0 operational BIN
s none               -   - -           HK_data        timeout      TMP   0 anystate    BIN
s TMP_RunDown        -   - -           TMP_RunDown    CMD_ack      RTS   1 anystate    BIN 2 1=TP1_RunDown/2=TP2_RunDown        1
s none               -   - -           -              TMPdae_TXcmd sdtmp 0 operational BIN
s none               -   - -           HK_data        timeout      TMP   0 anystate    BIN
s TMP_SwitchMotor    -   - -           TPM_SwitchMotor CMD_ack     RTS   1 anystate    BIN 2 1=TP1_SwitchMotor/2=TP2_SwitchMotor 1
s none               -   - -            -             TMPdae_TXcmd sdtmp 0 operational BIN
s none               -   - -            HK_data       timeout      TMP   0 anystate    BIN
```

### 7.4 Automated Stimulation with External Commands

The command handler was stimulated with external commands generated by the Ethernet daemon (ethdae) and MIL-bus daemon (mildae). To initiate this automated stimulation the ethdae and mildae were declared for automated test stimulation in the easysystem.def file. Due to this request the toolset added lines to the comamnd procedure table (lines 1-3) which initiate a periodic activity with a (varying) period between 1.4s and 2.8s. By each periodic event the system function EaSyAutotestuser is called which injects a command out of the set of commands of the current user state set which - in this case - only consists of the state "cmdstate" and the related incoming command "telescience". When this state has been entered external comands are periodically sent to the command handler "cmdHandler" by the system function genTelescienceCommand. This function evaluates the automatically generated tables (see previous section 7.3) and selects randomly an external command. It checks the coverage of external commands and when a user-defined limit has been reached it terminates automatically the run.

At system termination a report is generated by each of the command injection processes which gives an overview on the number of injected commands, the number of available commands, the achieved mean injection rate, the minimum and maximum coverage of commands and how many commands have the minimum and maximum coverage.

```
s none              ethdae 1 anystate startsystem       nocmd         f 1 1 0   b localch noresource 0.0 0.0 0.0 ethdae     0 samestate    0  0   0   bin
s none              ethdae 1 anystate autotestuser      period        l 0 0 0   b localch noresource 1.4 2.0 2.8 ethdae     0 samestate    0  0   0   bin
s EaSyAutoTestuser  ethdae 1 anystate cycleautotestuser nocmd         o 1 1 0   b localch noresource 1.4 2.0 2.8 ethdae     0 samestate    0  0   0   bin
s ethdae_init       ethdae 1 anystate sub_initcmd       subinitreturn f 1 1 255 b ch1     fcu        100 150 200 sysinit    1 anyuserstate 80 100 120 bin
s none              ethdae 1 anystate sub_initcmd       nocmd         l 1 1 255 b localch fcu        100 150 200 ethdae     1 cmdstate     80 100 120 bin
u genTelescienceCmd ethdae 1 cmdstate telescience       cdh_receive   o 1 1 255 b ch1     fcu        100 150 200 cmdhandler 1 anyuserstate 80 100 120 bin
```

```
+---------------------------------------+
+           Injected Commands           |
+---------------------------------------+

Process:                                ethdae/1
Injected commands:                      1320
Available commands:                     149
Mean injection rate:                    1.012512/s
Minimum coverage:                       2
Maximum coverage:                       17
Number of commands at minimum coverage: 1
Number of commands at maximum coverage: 1
All commands have been sent once at least
```

### 7.5 Check of Timeout Conditions

Timeout conditions are specified by adding a command line which includes "timeout" as outgoing command and in the column "incoming command" the command on which the timeout condition is set. This information requests the system to monitor the reception of the expected repsonse.

In the case shown below there is a cyclic activity related to the incoming command "cycleTMP_hk_trigger" which initiates periodic acquisition of housekeeping data from the "serial daemon of the turbo pump" (sdtmp). The expected response is the command "hk_data" on which a timeout condition of 0.4 s is set (lines 1-2).

Lines 3-5 are related to normal reception of the HK data which are processed by two user-defined functions "TMP_get_telemetry" and "TMP_Supervision". Line 3 has automatically be added by the system in order to reset the timeout monitoring when the expected response is received. This feature is available by an option a user has to activate. It helps to prevent false alarms due to timeout conditions which have not been reset manually in case the expected response is received in time.

Line 6 is executed when the hk_data are not received in time. In this case due to the timeout the command excto_HK_data is issued which is then processed by TMP and causes a message to be sent to the "Error Message Logging" process (emlog).

```
s none               -    - -         CYCLETMP_trigger TMPdae_TXcmd f 1 1 255 B ch1  - 100 150 200 sdtmp 0 operational 80 100 120 BIN
s none               -    - -         HK_data          timeout      l 1 1 255 b ch1  - 0.4 0.4 0.4 TMP   0 anystate    80 100 120 BIN
s none               -    - -         HK_data          resettimer   f 1 1 255 b noch - 100 100 100 TMP   0 samestate   80 100 120 BIN
s TMP_get_telemetry  -    - -         HK_data          nocmd        s 1 1 255 b ch1  - 100 100 100 TMP   0 samestate   80 100 120 BIN
s TMP_Supervision    -    - -         HK_data          nocmd        l 1 1 255 b ch1  - 100 100 100 TMP   0 samestate   80 100 120 BIN

s TMP_HKtimeout      -    - anystate  exctoHK_data     EML_msg      o 1 1 255 b ch1  - 100 100 100 emlog 1 anystate    80 100 120 BIN
```

### 7.6 Coverage Analysis

A number of reports is available which allow to analyse the coverage and related properties. Such reports are:

- coverage of command lines
- a list of non-covered command lines
- coverage of states
- a list of non-covered states
- executed state transitions
- exception report
- error injection report

The following figures where obtained for the "operational case", i.e. the external commands were automatically injected until for all external commands a minimum coverage of 2 occured at least. In this case an "effective" coverage of 100% for command lines and states is expected. "Effective coverage" means that for some cases as discussed below a coverage of less than 100% is acceptable. This may happen e.g. if lines deal with exception handling.

When an effective coverage of 100% is achieved without error messages the system is considered to behave correctly.

Coverage is measured for each of the instances of a process (if more than one) and for all instances, i.e. the sum of activities of every instance.

#### 7.6.1 Coverage of Command Lines

Coverage of command lines is defined as the ratio of command lines which are executed once at least and the total number of command lines of a process (not counting system specific system command lines like such related to error injection).

For each process, for each instance of a process and the summary figure for all instances of a process the coverage of command lines is given. Finally, an average figure for the whole system is derived.

Such command lines which were not executed are listed separately so that a user can decide whether this is acceptable or not and he gets a hint what is wrong in the system.

The following pages give all the figures related to command line coverage for the 37 MSL processes.

```
+-----------------------------------------------+
+           Coverage of Command Lines           |
+-----------------------------------------------+

Coverage of command lines: acq_hdl/1      100.000000 % for 3   lines
Coverage of command lines: acq_hdl/2      100.000000 % for 3   lines
Coverage of command lines: acq_hdl/all    100.000000 % for 3   lines

Coverage of command lines: anadae/1       100.000000 % for 4   lines
Coverage of command lines: anadae/2       100.000000 % for 4   lines
Coverage of command lines: anadae/all     100.000000 % for 4   lines

Coverage of command lines: cfv/1          100.000000 % for 7   lines
```

```
Coverage of command lines: cfv/all         100.000000 % for 7   lines

Coverage of command lines: cmdhandler/1    100.000000 % for 8   lines
Coverage of command lines: cmdhandler/all  100.000000 % for 8   lines

Coverage of command lines: digdae/1        87.500000  % for 64  lines
Coverage of command lines: digdae/2        18.750000  % for 64  lines
Coverage of command lines: digdae/all      100.000000 % for 64  lines

Coverage of command lines: ede/1           100.000000 % for 22  lines
Coverage of command lines: ede/all         100.000000 % for 22  lines

Coverage of command lines: emlog/1         100.000000 % for 3   lines
Coverage of command lines: emlog/all       100.000000 % for 3   lines

Coverage of command lines: ethdae/1        100.000000 % for 6   lines
Coverage of command lines: ethdae/all      100.000000 % for 6   lines

Coverage of command lines: fdr/1           100.000000 % for 10  lines
Coverage of command lines: fdr/all         100.000000 % for 10  lines

Coverage of command lines: hc_cychdl/1     100.000000 % for 4   lines
Coverage of command lines: hc_cychdl/all   100.000000 % for 4   lines

Coverage of command lines: hc_exe/1        100.000000 % for 14  lines
Coverage of command lines: hc_exe/all      100.000000 % for 14  lines

Coverage of command lines: mfg/1           95.454544  % for 22  lines
Coverage of command lines: mfg/all         95.454544  % for 22  lines

Coverage of command lines: mildae/1        100.000000 % for 6   lines
Coverage of command lines: mildae/all      100.000000 % for 6   lines

Coverage of command lines: mm_dae/1        100.000000 % for 2   lines
Coverage of command lines: mm_dae/all      100.000000 % for 2   lines

Coverage of command lines: ms__sv/1        100.000000 % for 10  lines
Coverage of command lines: ms__sv/all      100.000000 % for 10  lines

Coverage of command lines: msp/1           100.000000 % for 63  lines
Coverage of command lines: msp/all         100.000000 % for 63  lines

Coverage of command lines: oft/1           95.454544  % for 22  lines
Coverage of command lines: oft/all         95.454544  % for 22  lines

Coverage of command lines: opcexe/1        100.000000 % for 15  lines
Coverage of command lines: opcexe/all      100.000000 % for 15  lines

Coverage of command lines: opcsys/1        100.000000 % for 9   lines
Coverage of command lines: opcsys/all      100.000000 % for 9   lines

Coverage of command lines: pcs/1           100.000000 % for 3   lines
Coverage of command lines: pcs/all         100.000000 % for 3   lines

Coverage of command lines: qde/1           100.000000 % for 4   lines
Coverage of command lines: qde/all         100.000000 % for 4   lines

Coverage of command lines: rgv/1           100.000000 % for 3   lines
Coverage of command lines: rgv/all         100.000000 % for 3   lines

Coverage of command lines: scf/1           100.000000 % for 16  lines
Coverage of command lines: scf/all         100.000000 % for 16  lines
```

```
Coverage of command lines: scr/1          100.000000 % for 3   lines
Coverage of command lines: scr/all        100.000000 % for 3   lines

Coverage of command lines: sdmsp/1        100.000000 % for 7   lines
Coverage of command lines: sdmsp/all      100.000000 % for 7   lines

Coverage of command lines: sdpyr/1        100.000000 % for 4   lines
Coverage of command lines: sdpyr/all      100.000000 % for 4   lines

Coverage of command lines: sdt1/1         100.000000 % for 4   lines
Coverage of command lines: sdt1/2         100.000000 % for 4   lines
Coverage of command lines: sdt1/3         100.000000 % for 4   lines
Coverage of command lines: sdt1/4         100.000000 % for 4   lines
Coverage of command lines: sdt1/all       100.000000 % for 4   lines

Coverage of command lines: sdtmp/1        100.000000 % for 4   lines
Coverage of command lines: sdtmp/2        100.000000 % for 4   lines
Coverage of command lines: sdtmp/all      100.000000 % for 4   lines

Coverage of command lines: sysinit/1      99.295776  % for 142 lines
Coverage of command lines: sysinit/all    99.295776  % for 142 lines

Coverage of command lines: tmp/1          96.551720  % for 29  lines
Coverage of command lines: tmp/2          96.551720  % for 29  lines
Coverage of command lines: tmp/all        96.551720  % for 29  lines

Coverage of command lines: tmy_cychdl/1   100.000000 % for 4   lines
Coverage of command lines: tmy_cychdl/all 100.000000 % for 4   lines

Coverage of command lines: tmy_reqhdl/1   100.000000 % for 9   lines
Coverage of command lines: tmy_reqhdl/all 100.000000 % for 9   lines

Coverage of command lines: usd/1          100.000000 % for 22  lines
Coverage of command lines: usd/all        100.000000 % for 22  lines

Coverage of command lines: vgs/1          100.000000 % for 4   lines
Coverage of command lines: vgs/all        100.000000 % for 4   lines

Coverage of command lines: vgs_sv/1       100.000000 % for 2   lines
Coverage of command lines: vgs_sv/all     100.000000 % for 2   lines

Coverage of command lines: wpa/1          100.000000 % for 6   lines
Coverage of command lines: wpa/all        100.000000 % for 6   lines

Coverage of command lines: wpp/1          100.000000 % for 22  lines
Coverage of command lines: wpp/all        100.000000 % for 22  lines
```

A list of the non-executed command lines follows.

All of the listed non-covered command lines are identified as "exceptional cases" so that a full coverage of command lines is achieved.

In case of the digital daemon "digdae" the process is capable to execute all commands but there are limitations due to the avaialble hardware because instance 1 is running on FCU and instance 2 on PSU. Such limitations are reflected by the specification of external commands where for the command lines as listed below only reference is made to one of the instances.

The essential point here is that a command line is executed for one of both instances at least, which is confirmed by the fact that a non-executed command line is not listed for digdae/all and the coverage figure of digdae/all is 100%.

```
+-------------------------------------------+
+          Non-Executed Command Lines       |
+-------------------------------------------+

line 33  digdae(1)/anyState/led_saflamp
line 40  digdae(1)/anyState/mfg_switchdevice
line 55  digdae(1)/anyState/qde_setparams
line 56  digdae(1)/anyState/qde_start
line 57  digdae(1)/anyState/qde_stop
line 58  digdae(1)/anyState/hcs_setparams
line 59  digdae(1)/anyState/pcs_setpulse
line 64  digdae(1)/anyState/psu_resetsafetyinhibit
line 3   digdae(2)/anyState/cfd_setspeed
line 4   digdae(2)/anyState/cfd_setramp
line 5   digdae(2)/anyState/cfd_ramptospeed
line 6   digdae(2)/anyState/cfd_rampintimetospeed
line 7   digdae(2)/anyState/cfd_movewithspeedtoposition
line 8   digdae(2)/anyState/cfd_moveintimetoposition
line 9   digdae(2)/anyState/cfd_stopdrive
line 10  digdae(2)/anyState/usd_switchdevice
line 11  digdae(2)/anyState/tp1_switchdevice
line 12  digdae(2)/anyState/tp2_switchdevice
line 13  digdae(2)/anyState/vid_switchdevice
line 14  digdae(2)/anyState/acc_switchdevice
line 15  digdae(2)/anyState/ms__switchdevice
line 16  digdae(2)/anyState/cfd_switchclutch
line 17  digdae(2)/anyState/ede_switchdevice
line 18  digdae(2)/anyState/vgs_switchventvalve
line 19  digdae(2)/anyState/vgs_switchsol_valve1
line 20  digdae(2)/anyState/vgs_switchsol_valve2
line 21  digdae(2)/anyState/vgs_switchsol_valve6
line 22  digdae(2)/anyState/qde_switchdevice
line 23  digdae(2)/anyState/cf__switchdoorbolt
line 24  digdae(2)/anyState/st1_switchdevice
line 25  digdae(2)/anyState/st2_switchdevice
line 26  digdae(2)/anyState/oft_switchdevice_on
line 27  digdae(2)/anyState/oft_switchdevice_off
line 28  digdae(2)/anyState/led_normal
line 29  digdae(2)/anyState/led_processnom
line 30  digdae(2)/anyState/led_doorunlock
line 31  digdae(2)/anyState/led_heaterena
line 32  digdae(2)/anyState/led_testreprog
line 34  digdae(2)/anyState/led_test_start
line 35  digdae(2)/anyState/cfv_motorvalve_full
line 36  digdae(2)/anyState/cfv_motorvalve_partial
line 37  digdae(2)/anyState/vgs_motorvalve_full
line 38  digdae(2)/anyState/pd1_resetcurrentbreaker
line 39  digdae(2)/anyState/pd2_resetcurrentbreaker
line 43  digdae(2)/anyState/fud_setposition
line 44  digdae(2)/anyState/fud_setspeed
line 45  digdae(2)/anyState/fud_start
line 46  digdae(2)/anyState/fud_stop
line 47  digdae(2)/anyState/rgv_setposition
line 48  digdae(2)/anyState/rgv_setspeed
line 49  digdae(2)/anyState/rgv_start
line 50  digdae(2)/anyState/rgv_stop
line 51  digdae(2)/anyState/shc_setposition
line 52  digdae(2)/anyState/shc_setspeed
line 53  digdae(2)/anyState/shc_start
line 54  digdae(2)/anyState/shc_stop
line 60  digdae(2)/anyState/wpa_switchdevice
line 61  digdae(2)/anyState/dc1_switchsensorpower
```

```
line 62  digdae(2)/anyState/vgs_switchpressuresensor
line 63  digdae(2)/anyState/sca_switchpiranigauge
```

The following non-executed command lines represent timeout conditions. The timeout values have been selected such that most of the timeouts expire at least once. For the timeouts listed below no expiration has been observed.

For the sysinit process it is not desirable to get a timeout because then the initialisation procedure is aborted. It is similar for the other exceptions because they would occur during the initialisation steps. If desired other values may be chosen to enforce execution of such lines but then other lines will not be executed due to abort of system initialisation.

```
line 22  mfg(1)/anyState/exctodaemonack
line 22  mfg(all)/anyState/exctodaemonack

line 21  oft(1)/anyState/exctodaemonack
line 21  oft(all)/anyState/exctodaemonack

line 142 sysinit(1)/anyState/exctosubinitreturn
line 142 sysinit(all)/anyState/exctosubinitreturn

line 29  tmp(1)/anyState/exctodaemonack
line 29  tmp(2)/anyState/exctodaemonack
line 29  tmp(all)/anyState/exctodaemonack
```

### 7.6.2 Coverage of States

Coverage of states is defined as the ratio of the sum of (user states + anystate if used + asyncstate if used) which are executed once at least and the total number of the set of states of a process.

For each process, for each instance of a process and the summary figure for all instances of a process the coverage of states is given. Finally, an average figure for the whole system is derived.

Such states which were not executed are listed separately so that a user can decide whether this is acceptable or not and he gets a hint what is wrong in the system.

The following pages give all the figures related to state coverage.

For all processes and each of their instances a coverage figure of 100% is achieved.

```
          +---------------------------------------+
          +           Coverage of States          |
          +---------------------------------------+

Coverage of defined states: acq_hdl/1      100.000000 % for 1  states
Coverage of defined states: acq_hdl/2      100.000000 % for 1  states
Coverage of defined states: acq_hdl/all    100.000000 % for 1  states

Coverage of defined states: anadae/1       100.000000 % for 1  states
Coverage of defined states: anadae/2       100.000000 % for 1  states
Coverage of defined states: anadae/all     100.000000 % for 1  states

Coverage of defined states: cfv/1          100.000000 % for 1  states
Coverage of defined states: cfv/all        100.000000 % for 1  states

Coverage of defined states: cmdhandler/1   100.000000 % for 1  states
Coverage of defined states: cmdhandler/all 100.000000 % for 1  states
```

```
Coverage of defined states: digdae/1       100.000000 % for 1  states
Coverage of defined states: digdae/2       100.000000 % for 1  states
Coverage of defined states: digdae/all     100.000000 % for 1  states

Coverage of defined states: ede/1          100.000000 % for 5  states
Coverage of defined states: ede/all        100.000000 % for 5  states

Coverage of defined states: emlog/1        100.000000 % for 1  states
Coverage of defined states: emlog/all      100.000000 % for 1  states

Coverage of defined states: ethdae/1       100.000000 % for 2  states
Coverage of defined states: ethdae/all     100.000000 % for 2  states

Coverage of defined states: fdr/1          100.000000 % for 2  states
Coverage of defined states: fdr/all        100.000000 % for 2  states

Coverage of defined states: hc_cychdl/1    100.000000 % for 1  states
Coverage of defined states: hc_cychdl/all  100.000000 % for 1  states

Coverage of defined states: hc_exe/1       100.000000 % for 1  states
Coverage of defined states: hc_exe/all     100.000000 % for 1  states

Coverage of defined states: mfg/1          100.000000 % for 5  states
Coverage of defined states: mfg/all        100.000000 % for 5  states

Coverage of defined states: mildae/1       100.000000 % for 2  states
Coverage of defined states: mildae/all     100.000000 % for 2  states

Coverage of defined states: mm_dae/1       100.000000 % for 1  states
Coverage of defined states: mm_dae/all     100.000000 % for 1  states

Coverage of defined states: ms__sv/1       100.000000 % for 1  states
Coverage of defined states: ms__sv/all     100.000000 % for 1  states

Coverage of defined states: msp/1          100.000000 % for 5  states
Coverage of defined states: msp/all        100.000000 % for 5  states

Coverage of defined states: oft/1          100.000000 % for 5  states
Coverage of defined states: oft/all        100.000000 % for 5  states

Coverage of defined states: opcexe/1       100.000000 % for 1  states
Coverage of defined states: opcexe/all     100.000000 % for 1  states

Coverage of defined states: opcsys/1       100.000000 % for 1  states
Coverage of defined states: opcsys/all     100.000000 % for 1  states

Coverage of defined states: pcs/1          100.000000 % for 1  states
Coverage of defined states: pcs/all        100.000000 % for 1  states

Coverage of defined states: qde/1          100.000000 % for 1  states
Coverage of defined states: qde/all        100.000000 % for 1  states

Coverage of defined states: rgv/1          100.000000 % for 1  states
Coverage of defined states: rgv/all        100.000000 % for 1  states

Coverage of defined states: scf/1          100.000000 % for 1  states
Coverage of defined states: scf/all        100.000000 % for 1  states

Coverage of defined states: scr/1          100.000000 % for 1  states
Coverage of defined states: scr/all        100.000000 % for 1  states

Coverage of defined states: sdmsp/1        100.000000 % for 3  states
```

```
Coverage of defined states: sdmsp/all      100.000000 % for 3  states

Coverage of defined states: sdpyr/1        100.000000 % for 3  states
Coverage of defined states: sdpyr/all      100.000000 % for 3  states

Coverage of defined states: sdt1/1         100.000000 % for 3  states
Coverage of defined states: sdt1/2         100.000000 % for 2  states
Coverage of defined states: sdt1/3         100.000000 % for 2  states
Coverage of defined states: sdt1/4         100.000000 % for 2  states
Coverage of defined states: sdt1/all       100.000000 % for 3  states

Coverage of defined states: sdtmp/1        100.000000 % for 3  states
Coverage of defined states: sdtmp/2        100.000000 % for 2  states
Coverage of defined states: sdtmp/all      100.000000 % for 3  states

Coverage of defined states: sysinit/1      100.000000 % for 36 states
Coverage of defined states: sysinit/all    100.000000 % for 36 states

Coverage of defined states: tmp/1          100.000000 % for 5  states
Coverage of defined states: tmp/2          100.000000 % for 5  states
Coverage of defined states: tmp/all        100.000000 % for 5  states

Coverage of defined states: tmy_cychdl/1   100.000000 % for 1  states
Coverage of defined states: tmy_cychdl/all 100.000000 % for 1  states

Coverage of defined states: tmy_reqhdl/1   100.000000 % for 1  states
Coverage of defined states: tmy_reqhdl/all 100.000000 % for 1  states

Coverage of defined states: usd/1          100.000000 % for 5  states
Coverage of defined states: usd/all        100.000000 % for 5  states

Coverage of defined states: vgs/1          100.000000 % for 1  states
Coverage of defined states: vgs/all        100.000000 % for 1  states

Coverage of defined states: vgs_sv/1       100.000000 % for 1  states
Coverage of defined states: vgs_sv/all     100.000000 % for 1  states

Coverage of defined states: wpa/1          100.000000 % for 1  states
Coverage of defined states: wpa/all        100.000000 % for 1  states

Coverage of defined states: wpp/1          100.000000 % for 5  states
Coverage of defined states: wpp/all        100.000000 % for 5  states
```

As mentionned already above all states have been covered therefore the following list of non-covered states is empty

```
+-------------------------------------+
+           Non-covered States        |
+-------------------------------------+
```

**"Empty list"**

### 7.6.3 State Transitions

The report on state transitions is "informal" because there is no formal constraint on coverage of state transitions. It depends on the system design whether all or only some state transtions are executed. From a verification point of view the coverage of command lines and states is formal in the sense that an effective coverage of 100% shall be achieved. If such figures are achieved it is confirmed implicitly that all relevant state transitions have been executed.

It may even occur - as it is true for most processes of MSL as shown by the report below - that the system remains in the initial state and no state transitions really occurs. So the following report shall help an engineer to understand what is going on, it may give him some indication to change a command line. But by the presence or absence of state transitions no direct conclusions on the correctness or completeness are possible.

Most of the state transtions listed below are related to system and process initialisation.

```
+---------------------------------------+
+           State Transitions           |
+---------------------------------------+

  0 state transitions for acq_hdl/1
  --------------------------------

  0 state transitions for acq_hdl/2
  --------------------------------

  0 state transitions for acq_hdl/all
  --------------------------------

  0 state transitions for anadae/1
  --------------------------------

  0 state transitions for anadae/2
  --------------------------------

  0 state transitions for anadae/all
  --------------------------------

  0 state transitions for cfv/1
  --------------------------------

  0 state transitions for cfv/all
  --------------------------------

  0 state transitions for cmdhandler/1
  --------------------------------

  0 state transitions for cmdhandler/all
  --------------------------------

  0 state transitions for digdae/1
  --------------------------------

  0 state transitions for digdae/2
  --------------------------------

  0 state transitions for digdae/all
  --------------------------------
```

```
pre_init  -> init        1 for ede/1
init      -> init_done   1 for ede/1
init_done -> operational 1 for ede/1

  3 state transitions for ede/1
  -------------------------------

pre_init  -> init        1 for ede/all
init      -> init_done   1 for ede/all
init_done -> operational 1 for ede/all

  3 state transitions for ede/all
  -------------------------------

  0 state transitions for emlog/1
  -------------------------------


  0 state transitions for emlog/all
  -------------------------------

anyState -> cmdstate 1 for ethdae/1

  1 state transitions for ethdae/1
  -------------------------------

anyState -> cmdstate 1 for ethdae/all

  1 state transitions for ethdae/all
  -------------------------------

anyState -> supervise 10 for fdr/1

  1 state transitions for fdr/1
  -------------------------------

anyState -> supervise 10 for fdr/all

  1 state transitions for fdr/all
  -------------------------------

  0 state transitions for hc_cychdl/1
  -------------------------------

  0 state transitions for hc_cychdl/all
  -------------------------------

  0 state transitions for hc_exe/1
  -------------------------------

  0 state transitions for hc_exe/all
  -------------------------------

pre_init  -> init        1 for mfg/1
init      -> init_done   1 for mfg/1
init_done -> operational 1 for mfg/1

  3 state transitions for mfg/1
  -------------------------------

pre_init  -> init        1 for mfg/all
```

```
init      -> init_done   1 for mfg/all
init_done -> operational 1 for mfg/all

  3 state transitions for mfg/all
  --------------------------------

anyState -> cmdstate 1 for mildae/1

  1 state transitions for mildae/1
  --------------------------------

anyState -> cmdstate 1 for mildae/all

  1 state transitions for mildae/all
  --------------------------------

  0 state transitions for mm_dae/1
  --------------------------------


  0 state transitions for mm_dae/all
  --------------------------------

  0 state transitions for ms__sv/1
  --------------------------------

  0 state transitions for ms__sv/all
  --------------------------------

pre_init  -> init        1 for msp/1
init      -> init_done   1 for msp/1
init_done -> operational 1 for msp/1

  3 state transitions for msp/1
  --------------------------------

pre_init  -> init        1 for msp/all
init      -> init_done   1 for msp/all
init_done -> operational 1 for msp/all

  3 state transitions for msp/all
  --------------------------------

pre_init  -> init        1 for oft/1
init      -> init_done   1 for oft/1
init_done -> operational 1 for oft/1

  3 state transitions for oft/1
  --------------------------------

pre_init  -> init        1 for oft/all
init      -> init_done   1 for oft/all
init_done -> operational 1 for oft/all

  3 state transitions for oft/all
  --------------------------------

  0 state transitions for opcexe/1
  --------------------------------

  0 state transitions for opcexe/all
  --------------------------------
```

```
  0 state transitions for opcsys/1
  --------------------------------

  0 state transitions for opcsys/all
  --------------------------------

  0 state transitions for pcs/1
  --------------------------------

  0 state transitions for pcs/all
  --------------------------------

  0 state transitions for qde/1
  --------------------------------

  0 state transitions for qde/all
  --------------------------------

  0 state transitions for rgv/1
  --------------------------------

  0 state transitions for rgv/all
  --------------------------------

  0 state transitions for scf/1
  --------------------------------

  0 state transitions for scf/all
  --------------------------------

  0 state transitions for scr/1
  --------------------------------

  0 state transitions for scr/all
  --------------------------------

init -> operational 1 for sdmsp/1

  1 state transitions for sdmsp/1
  --------------------------------

init -> operational 1 for sdmsp/all

  1 state transitions for sdmsp/all
  --------------------------------

init -> operational 1 for sdpyr/1

  1 state transitions for sdpyr/1
  --------------------------------

init -> operational 1 for sdpyr/all

  1 state transitions for sdpyr/all
  --------------------------------

init -> operational 1 for sdt1/1

  1 state transitions for sdt1/1
  --------------------------------
```

```
init -> operational 1 for sdt1/2

  1 state transitions for sdt1/2
  ------------------------------

init -> operational 1 for sdt1/3

  1 state transitions for sdt1/3
  ------------------------------

init -> operational 1 for sdt1/4

  1 state transitions for sdt1/4
  ------------------------------

init -> operational 4 for sdt1/all

  1 state transitions for sdt1/all
  --------------------------------

init -> operational 1 for sdtmp/1

  1 state transitions for sdtmp/1
  -------------------------------

init -> operational 1 for sdtmp/2

  1 state transitions for sdtmp/2
  -------------------------------

init -> operational 2 for sdtmp/all

  1 state transitions for sdtmp/all
  ---------------------------------

anyState -> state0  1 for sysinit/1
state0   -> state1  1 for sysinit/1
state1   -> state2  1 for sysinit/1
state2   -> state3  1 for sysinit/1
state3   -> state4  1 for sysinit/1
state4   -> state5  1 for sysinit/1
state5   -> state6  1 for sysinit/1
state6   -> state7  1 for sysinit/1
state7   -> state8  1 for sysinit/1
state8   -> state9  1 for sysinit/1
state9   -> state10 1 for sysinit/1
state10  -> state11 1 for sysinit/1
state11  -> state12 1 for sysinit/1
state12  -> state13 1 for sysinit/1
state13  -> state14 1 for sysinit/1
state14  -> state15 1 for sysinit/1
state15  -> state16 1 for sysinit/1
state16  -> state17 1 for sysinit/1
state17  -> state18 1 for sysinit/1
state18  -> state19 1 for sysinit/1
state19  -> state20 1 for sysinit/1
state20  -> state21 1 for sysinit/1
state21  -> state22 1 for sysinit/1
state22  -> state23 1 for sysinit/1
state23  -> state24 1 for sysinit/1
state24  -> state25 1 for sysinit/1
state25  -> state26 1 for sysinit/1
```

```
state26  -> state27 1 for sysinit/1
state27  -> state28 1 for sysinit/1
state28  -> state29 1 for sysinit/1
state29  -> state30 1 for sysinit/1
state30  -> state31 1 for sysinit/1
state31  -> state32 1 for sysinit/1
state32  -> state33 1 for sysinit/1
state33  -> state34 1 for sysinit/1

  35 state transitions for sysinit/1
  ---------------------------------

  35 state transitions for sysinit/all
  ---------------------------------

pre_init  -> init        1 for tmp/1
init      -> init_done   1 for tmp/1
init_done -> operational 1 for tmp/1

  3 state transitions for tmp/1
  ---------------------------------

pre_init  -> init        1 for tmp/2
init      -> init_done   1 for tmp/2
init_done -> operational 1 for tmp/2

  3 state transitions for tmp/2
  ---------------------------------

pre_init  -> init        2 for tmp/all
init      -> init_done   2 for tmp/all
init_done -> operational 2 for tmp/all

  3 state transitions for tmp/all
  ---------------------------------

  0 state transitions for tmy_cychdl/1
  ---------------------------------

  0 state transitions for tmy_cychdl/all
  ---------------------------------

  0 state transitions for tmy_reqhdl/1
  ---------------------------------

  0 state transitions for tmy_reqhdl/all
  ---------------------------------

pre_init  -> init        1 for usd/1
init      -> init_done   1 for usd/1
init_done -> operational 1 for usd/1

  3 state transitions for usd/1
  ---------------------------------

pre_init  -> init        1 for usd/all
init      -> init_done   1 for usd/all
init_done -> operational 1 for usd/all

  3 state transitions for usd/all
  ---------------------------------
```

```
  0 state transitions for vgs/1
  -------------------------------

  0 state transitions for vgs/all
  -------------------------------

  0 state transitions for vgs_sv/1
  -------------------------------

  0 state transitions for vgs_sv/all
  -------------------------------

  0 state transitions for wpa/1
  -------------------------------

  0 state transitions for wpa/all
  -------------------------------


pre_init  -> init        1 for wpp/1
init      -> init_done   1 for wpp/1
init_done -> operational 1 for wpp/1

  3 state transitions for wpp/1
  -------------------------------

pre_init  -> init        1 for wpp/all
init      -> init_done   1 for wpp/all
init_done -> operational 1 for wpp/all

  3 state transitions for wpp/all
  -------------------------------
```

### 7.6.4 Exception Report

For each command line the number of exceptions is traced which occur when the comand line is executed. There are a number of exceptions and it depends on the contents of the command line what the reason of an exception is. Following exceptions may occur: timeout, overrun of a cyclic activity, deadline exceeded.

Exceptions also may occur due to error injection if this feature is activated.

The exceptions listed below are related to timeout conditions, they are desired in order to get a higher coverage, a coverage of such command lines which handle expiration of timeouts (command prefix is "excto").

If the exception report is not empty it usually indicates a problem. By the listed command lines an engineer shall get the information THAT and WHERE an exception occured and hence where a problem exists or may exist.

```
+-------------------------------------------------------+
+           Exception Report                            |
+-------------------------------------------------------+


line 8  cmdhandler(all)/anyState/cmd_ack 44  exceptions

line 2  ede(all)/pre_init/daemonack      1   exceptions
line 6  ede(all)/init/hk_data            37  exceptions

line 6  mfg(all)/init/hk_data            25  exceptions

line 5  ms__sv(all)/anyState/daemonack   15  exceptions

line 2  msp(all)/pre_init/daemonack      100 exceptions
line 47 msp(all)/operational/ms_specdata 161 exceptions
line 51 msp(all)/operational/ms_hkdata   78  exceptions
line 56 msp(all)/operational/ms_trpdata  21  exceptions

line 6  oft(all)/init/hk_data            22  exceptions

line 6  tmp(1)/init/hk_data              22  exceptions
line 6  tmp(2)/init/hk_data              2   exceptions
line 6  tmp(all)/init/hk_data            24  exceptions

line 2  usd(all)/pre_init/daemonack      1   exceptions
line 6  usd(all)/init/hk_data            19  exceptions

line 2  wpp(all)/pre_init/daemonack      1   exceptions
line 6  wpp(all)/init/hk_data            29  exceptions
```

### 7.6.5 Error Injection Report

The report on error injection related to the operational case is empty because error injection was not requested for obvious reasons. Error injection will be treated separately.

```
+-------------------------------------------------+
+           Error Injection Report                |
+-------------------------------------------------+
```

**Empty List**

## 7.7 Performance Analysis

For performance analysis a number of reports are generated:

- CPU utilisation

  for all allocated nodes / CPU's
- timer report
- load calibration report
- network utilisation report
- response time report
- command buffer report

These reports shall give information about utilisation of system resources and indicate potential problems with performance.

### 7.7.1 CPU Utilisation

Utilisation of CPU's is measured in different ways depending on whether the system is in simulation or real-time mode. In simulation mode the provided figures for a CPU or network channel are taken into account. In real-time mode - which was selected for the execution of the operational case - either the figures provided by the system - as it is the case for UNIX environments - or equivalent figures were provided by ISG utilities.

The figures given below take benefit of the UNIX capabilities.

The multi-processor system (two CPU's FCU and PSU) was executed on a single host. But the consumption for each node and each communication line was tracked separately.

As the command procedure table includes the information which instance of a process executes on which node / CPU such allocation is possible. Also, an estimation of the expected load of each CPU is possible by assigning the figures of instances to the associated CPU.

Same is true for the network traffic.

The figures on the next pages give the following information:

- by the set of types of lines 1 - 8
  - -- line 1: the duration of the execution for each process
  - -- line 2: the time consumed by the sequential part of all the instances of a process excluding time consumed for interrupts and I/O handling
  - -- line 3: the same figure as for line 2 but for a dedicated instance for which the number is given by column 3
  - -- line 4: the time consumed by all the instances of a process for interrupts and I/O handling
  - -- line 5: the same figure as for line 4 but for a dedicated instance for which the number is given by column 3
  - -- line 6: the total time consumed by the all the instances of a process (sum of lines 2 and 4)
  - -- line 7: the same figure as for line 6 but for a dedicated instance for which the number is given by column 3

-- line 7a: the corrected total time for instance by comparison of line 6 and 8

-- line 8: the total time a process has consumed as provided by the OS at process termination (if available)

- by the columns

-- col 1 the type of a line

-- col 2 the name of the process

-- col 3 the instance number of a process or "all" for all instances

-- col 4 the consumed time in seconds

-- col 5 the percentage of consumed time w.r.t. to the process (type)

-- col 6 the percentage of consumed time w.r.t. to the total time consumed by the system

-- col 7 the contribution to CPU utilisation (consumed time over duration of execution)

-- col 8 a comment on the line type

-- col 9 the CPU to which this time consumption is charged

Finally, the last three lines give the total system utilisation.

If the system execution mode is "ONETARGETONLY" an estimation for the different nodes is made by the following lines otherwise the measured utilisation figures for each CPU are precisely given. Consumption of common parts like the timer process is assigned to each CPU according to the percentage each CPU contributes to the whole utilisation figure in case of "ONETARGETONLY" mode.

The current utilisation figures do not allow a precise conclusion whether the resources on the targets will be exceeded or not. Such a conclusion can be made when the system is executed the first time on the real target which is expected for the next two weeks[5].

The current user-defined functions do not include the real functionality (except for the command dispatcher), but they include a lot of instrumentation for report generation, test stimulation and input generation for the graphical tools (MSC's and timing diagrams tools). This causes a lot of resource consumption, but it is still open whether this is larger or smaller than the real consumption (however, the feeling is currently: it is larger).

[5] This was the time expected when the draft final report was written. See the final section of chapter 8 for more information.

```
+-------------------------------------------------------+
+           Evaluation of CPU Utilisation               |
+-------------------------------------------------------+


  PROCMIN=sdmsp MINDUR=1366.84 PROCMAX=hc_cychdl MAXDUR=1377.99


1  RSIM       all 1369.83s -       -      -       duration
2  RSIM       all 0.10s    2.31%   0.03%  0.007%  tot_sequProg_busyTime              -
3  RSIM       1   0.10s    2.31%   0.03%  0.007%  instance_sequProg_busyTime
4  RSIM       all 1.89s    43.75%  0.54%  0.138%  tot_asyncIO_busyTime               -
5  RSIM       1   1.89s    43.75%  0.54%  0.138%  instance_asyncIO_busyTime
6  RSIM       all 1.99s    46.06%  0.57%  0.146%  tot_process_busyTime               -
7  RSIM       1   1.99s    46.06%  0.57%  0.146%  instance_busyTime
7a RSIM       1   4.32s    100.00% 1.24%  0.316%  tot_busyTime(corrected)
8  RSIM       all 4.32s    100.00% 1.24%  0.316%  tot_busyTime(system)               -

1  acq_hdl    all 1372.18s -       -      -       duration
2  acq_hdl    all 7.30s    66.30%  2.09%  0.534%  tot_sequProg_busyTime              -
3  acq_hdl    1   7.30s    66.30%  2.09%  0.534%  instance_sequProg_busyTime         FCU
3  acq_hdl    2   0.00s    0.00%   0.00%  0.000%  instance_sequProg_busyTime         PSU
4  acq_hdl    all 3.45s    31.34%  0.99%  0.252%  tot_asyncIO_busyTime               -
5  acq_hdl    1   2.43s    22.07%  0.70%  0.178%  instance_asyncIO_busyTime          FCU
5  acq_hdl    2   1.02s    9.26%   0.29%  0.075%  instance_asyncIO_busyTime          PSU
6  acq_hdl    all 10.75s   97.64%  3.07%  0.786%  tot_process_busyTime               -
7  acq_hdl    1   9.73s    88.37%  2.78%  0.712%  instance_busyTime                  FCU
7a acq_hdl    1   9.97s    90.51%  2.85%  0.729%  tot_busyTime(corrected)            FCU
7  acq_hdl    2   1.02s    9.26%   0.29%  0.075%  instance_busyTime                  PSU
7a acq_hdl    2   1.04s    9.49%   0.30%  0.076%  tot_busyTime(corrected)            PSU
8  acq_hdl    all 11.01s   100.00% 3.15%  0.806%  tot_busyTime(system)               -

1  anadae     all 1370.11s -       -      -       duration
2  anadae     all 12.82s   84.62%  3.67%  0.938%  tot_sequProg_busyTime              -
3  anadae     1   12.82s   84.62%  3.67%  0.938%  instance_sequProg_busyTime         FCU
3  anadae     2   0.00s    0.00%   0.00%  0.000%  instance_sequProg_busyTime         PSU
4  anadae     all 2.20s    14.52%  0.63%  0.161%  tot_asyncIO_busyTime               -
5  anadae     1   1.55s    10.23%  0.44%  0.113%  instance_asyncIO_busyTime          FCU
```

```
5  anadae     2   0.65s    4.29%   0.19%  0.048%  instance_asyncIO_busyTime             PSU
6  anadae     all 15.02s   99.14%  4.30%  1.099%  tot_process_busyTime                  -
7  anadae     1   14.37s   94.85%  4.11%  1.051%  instance_busyTime                     FCU
7a anadae     1   14.49s   95.67%  4.15%  1.060%  tot_busyTime(corrected)               FCU
7  anadae     2   0.65s    4.29%   0.19%  0.048%  instance_busyTime                     PSU
7a anadae     2   0.66s    4.33%   0.19%  0.048%  tot_busyTime(corrected)               PSU
8  anadae     all 15.15s   100.00% 4.33%  1.108%  tot_busyTime(system)                  -

1  cfv        all 1368.96s -       -      -       duration
2  cfv        all 0.37s    46.25%  0.11%  0.027%  tot_sequProg_busyTime                 -
3  cfv        1   0.37s    46.25%  0.11%  0.027%  instance_sequProg_busyTime            FCU
4  cfv        all 0.33s    41.25%  0.09%  0.024%  tot_asyncIO_busyTime                  -
5  cfv        1   0.33s    41.25%  0.09%  0.024%  instance_asyncIO_busyTime             FCU
6  cfv        all 0.70s    87.50%  0.20%  0.051%  tot_process_busyTime                  -
7  cfv        1   0.70s    87.50%  0.20%  0.051%  instance_busyTime                     FCU
7a cfv        1   0.80s    100.00% 0.23%  0.059%  tot_busyTime(corrected)               FCU
8  cfv        all 0.80s    100.00% 0.23%  0.059%  tot_busyTime(system)                  -

1  cmdhandler all 1374.55s -       -      -       duration
2  cmdhandler all 16.29s   67.48%  4.66%  1.192%  tot_sequProg_busyTime                 -
3  cmdhandler 1   16.29s   67.48%  4.66%  1.192%  instance_sequProg_busyTime            FCU
4  cmdhandler all 7.30s    30.24%  2.09%  0.534%  tot_asyncIO_busyTime                  -
5  cmdhandler 1   7.30s    30.24%  2.09%  0.534%  instance_asyncIO_busyTime             FCU
6  cmdhandler all 23.59s   97.72%  6.75%  1.726%  tot_process_busyTime                  -
7  cmdhandler 1   23.59s   97.72%  6.75%  1.726%  instance_busyTime                     FCU
7a cmdhandler 1   24.14s   100.00% 6.90%  1.766%  tot_busyTime(corrected)               FCU
8  cmdhandler all 24.14s   100.00% 6.90%  1.766%  tot_busyTime(system)                  -

1  digdae     all 1375.59s -       -      -       duration
2  digdae     all 5.12s    63.84%  1.46%  0.375%  tot_sequProg_busyTime                 -
3  digdae     1   5.12s    63.84%  1.46%  0.375%  instance_sequProg_busyTime            FCU
3  digdae     2   0.00s    0.00%   0.00%  0.000%  instance_sequProg_busyTime            PSU
4  digdae     all 2.13s    26.56%  0.61%  0.156%  tot_asyncIO_busyTime                  -
5  digdae     1   1.92s    23.94%  0.55%  0.140%  instance_asyncIO_busyTime             FCU
5  digdae     2   0.21s    2.62%   0.06%  0.015%  instance_asyncIO_busyTime             PSU
6  digdae     all 7.25s    90.40%  2.07%  0.530%  tot_process_busyTime                  -
7  digdae     1   7.04s    87.78%  2.01%  0.515%  instance_busyTime                     FCU
7a digdae     1   7.79s    97.10%  2.23%  0.570%  tot_busyTime(corrected)               FCU
7  digdae     2   0.21s    2.62%   0.06%  0.015%  instance_busyTime                     PSU
```

```
7a digdae     2   0.23s    2.90%   0.07%  0.017%  tot_busyTime(corrected)              PSU
8  digdae     all 8.02s    100.00% 2.29%  0.587%  tot_busyTime(system)                 -

1  ede        all 1375.53s -       -      -       duration
2  ede        all 14.04s   73.51%  4.02%  1.027%  tot_sequProg_busyTime                -
3  ede        1   14.04s   73.51%  4.02%  1.027%  instance_sequProg_busyTime           FCU
4  ede        all 4.73s    24.76%  1.35%  0.346%  tot_asyncIO_busyTime                 -
5  ede        1   4.73s    24.76%  1.35%  0.346%  instance_asyncIO_busyTime            FCU
6  ede        all 18.77s   98.27%  5.37%  1.373%  tot_process_busyTime                 -
7  ede        1   18.77s   98.27%  5.37%  1.373%  instance_busyTime                    FCU
7a ede        1   19.10s   100.00% 5.46%  1.397%  tot_busyTime(corrected)              FCU
8  ede        all 19.10s   100.00% 5.46%  1.397%  tot_busyTime(system)                 -

1  emlog      all 1369.44s -       -      -       duration
2  emlog      all 2.22s    44.14%  0.63%  0.162%  tot_sequProg_busyTime                -
3  emlog      1   2.22s    44.14%  0.63%  0.162%  instance_sequProg_busyTime           FCU
4  emlog      all 2.67s    53.08%  0.76%  0.195%  tot_asyncIO_busyTime                 -
5  emlog      1   2.67s    53.08%  0.76%  0.195%  instance_asyncIO_busyTime            FCU
6  emlog      all 4.89s    97.22%  1.40%  0.358%  tot_process_busyTime                 -
7  emlog      1   4.89s    97.22%  1.40%  0.358%  instance_busyTime                    FCU
7a emlog      1   5.03s    100.00% 1.44%  0.368%  tot_busyTime(corrected)              FCU
8  emlog      all 5.03s    100.00% 1.44%  0.368%  tot_busyTime(system)                 -

1  ethdae     all 1372.92s -       -      -       duration
2  ethdae     all 16.79s   90.41%  4.80%  1.228%  tot_sequProg_busyTime                -
3  ethdae     1   16.79s   90.41%  4.80%  1.228%  instance_sequProg_busyTime           FCU
4  ethdae     all 1.59s    8.56%   0.45%  0.116%  tot_asyncIO_busyTime                 -
5  ethdae     1   1.59s    8.56%   0.45%  0.116%  instance_asyncIO_busyTime            FCU
6  ethdae     all 18.38s   98.98%  5.26%  1.345%  tot_process_busyTime                 -
7  ethdae     1   18.38s   98.98%  5.26%  1.345%  instance_busyTime                    FCU
7a ethdae     1   18.57s   100.00% 5.31%  1.359%  tot_busyTime(corrected)              FCU
8  ethdae     all 18.57s   100.00% 5.31%  1.359%  tot_busyTime(system)                 -

1  fdr        all 1369.53s -       -      -       duration
2  fdr        all 1.76s    61.11%  0.50%  0.129%  tot_sequProg_busyTime                -
3  fdr        1   1.76s    61.11%  0.50%  0.129%  instance_sequProg_busyTime           FCU
4  fdr        all 1.02s    35.42%  0.29%  0.075%  tot_asyncIO_busyTime                 -
5  fdr        1   1.02s    35.42%  0.29%  0.075%  instance_asyncIO_busyTime            FCU
6  fdr        all 2.78s    96.53%  0.80%  0.203%  tot_process_busyTime                 -
```

```
7  fdr        1   2.78s    96.53%  0.80%  0.203%  instance_busyTime                FCU
7a fdr        1   2.88s    100.00% 0.82%  0.211%  tot_busyTime(corrected)          FCU
8  fdr        all 2.88s    100.00% 0.82%  0.211%  tot_busyTime(system)             -

1  hc_cychdl  all 1377.99s -       -      -       duration
2  hc_cychdl  all 1.52s    57.79%  0.43%  0.111%  tot_sequProg_busyTime            -
3  hc_cychdl  1   1.52s    57.79%  0.43%  0.111%  instance_sequProg_busyTime       PSU
4  hc_cychdl  all 1.01s    38.40%  0.29%  0.074%  tot_asyncIO_busyTime             -
5  hc_cychdl  1   1.01s    38.40%  0.29%  0.074%  instance_asyncIO_busyTime        PSU
6  hc_cychdl  all 2.53s    96.20%  0.72%  0.185%  tot_process_busyTime             -
7  hc_cychdl  1   2.53s    96.20%  0.72%  0.185%  instance_busyTime                PSU
7a hc_cychdl  1   2.63s    100.00% 0.75%  0.192%  tot_busyTime(corrected)          PSU
8  hc_cychdl  all 2.63s    100.00% 0.75%  0.192%  tot_busyTime(system)             -

1  hc_exe     all 1372.28s -       -      -       duration
2  hc_exe     all 0.63s    47.01%  0.18%  0.046%  tot_sequProg_busyTime            -
3  hc_exe     1   0.63s    47.01%  0.18%  0.046%  instance_sequProg_busyTime       PSU
4  hc_exe     all 0.62s    46.27%  0.18%  0.045%  tot_asyncIO_busyTime             -
5  hc_exe     1   0.62s    46.27%  0.18%  0.045%  instance_asyncIO_busyTime        PSU
6  hc_exe     all 1.25s    93.28%  0.36%  0.091%  tot_process_busyTime             -
7  hc_exe     1   1.25s    93.28%  0.36%  0.091%  instance_busyTime                PSU
7a hc_exe     1   1.34s    100.00% 0.38%  0.098%  tot_busyTime(corrected)          PSU
8  hc_exe     all 1.34s    100.00% 0.38%  0.098%  tot_busyTime(system)             -

1  mfg        all 1376.34s -       -      -       duration
2  mfg        all 9.99s    79.79%  2.86%  0.731%  tot_sequProg_busyTime            -
3  mfg        1   9.99s    79.79%  2.86%  0.731%  instance_sequProg_busyTime       PSU
4  mfg        all 2.32s    18.53%  0.66%  0.170%  tot_asyncIO_busyTime             -
5  mfg        1   2.32s    18.53%  0.66%  0.170%  instance_asyncIO_busyTime        PSU
6  mfg        all 12.31s   98.32%  3.52%  0.901%  tot_process_busyTime             -
7  mfg        1   12.31s   98.32%  3.52%  0.901%  instance_busyTime                PSU
7a mfg        1   12.52s   100.00% 3.58%  0.916%  tot_busyTime(corrected)          PSU
8  mfg        all 12.52s   100.00% 3.58%  0.916%  tot_busyTime(system)             -

1  mildae     all 1370.95s -       -      -       duration
2  mildae     all 3.60s    90.00%  1.03%  0.263%  tot_sequProg_busyTime            -
3  mildae     1   3.60s    90.00%  1.03%  0.263%  instance_sequProg_busyTime       FCU
4  mildae     all 0.32s    8.00%   0.09%  0.023%  tot_asyncIO_busyTime             -
5  mildae     1   0.32s    8.00%   0.09%  0.023%  instance_asyncIO_busyTime        FCU
```

```
6  mildae     all 3.92s    98.00%  1.12%  0.287%  tot_process_busyTime                -
7  mildae     1   3.92s    98.00%  1.12%  0.287%  instance_busyTime                   FCU
7a mildae     1   4.00s    100.00% 1.14%  0.293%  tot_busyTime(corrected)             FCU
8  mildae     all 4.00s    100.00% 1.14%  0.293%  tot_busyTime(system)                -

1  mm_dae     all 1371.73s -       -      -       duration
2  mm_dae     all 0.06s    50.00%  0.02%  0.004%  tot_sequProg_busyTime               -
3  mm_dae     1   0.06s    50.00%  0.02%  0.004%  instance_sequProg_busyTime          FCU
4  mm_dae     all 0.00s    0.00%   0.00%  0.000%  tot_asyncIO_busyTime                -
5  mm_dae     1   0.00s    0.00%   0.00%  0.000%  instance_asyncIO_busyTime           FCU
6  mm_dae     all 0.06s    50.00%  0.02%  0.004%  tot_process_busyTime                -
7  mm_dae     1   0.06s    50.00%  0.02%  0.004%  instance_busyTime                   FCU
7a mm_dae     1   0.12s    100.00% 0.03%  0.009%  tot_busyTime(corrected)             FCU
8  mm_dae     all 0.12s    100.00% 0.03%  0.009%  tot_busyTime(system)                -

1  ms__sv     all 1372.21s -       -      -       duration
2  ms__sv     all 0.44s    60.27%  0.13%  0.032%  tot_sequProg_busyTime               -
3  ms__sv     1   0.44s    60.27%  0.13%  0.032%  instance_sequProg_busyTime          PSU
4  ms__sv     all 0.18s    24.66%  0.05%  0.013%  tot_asyncIO_busyTime                -
5  ms__sv     1   0.18s    24.66%  0.05%  0.013%  instance_asyncIO_busyTime           PSU
6  ms__sv     all 0.62s    84.93%  0.18%  0.045%  tot_process_busyTime                -
7  ms__sv     1   0.62s    84.93%  0.18%  0.045%  instance_busyTime                   PSU
7a ms__sv     1   0.73s    100.00% 0.21%  0.053%  tot_busyTime(corrected)             PSU
8  ms__sv     all 0.73s    100.00% 0.21%  0.053%  tot_busyTime(system)                -

1  msp        all 1371.10s -       -      -       duration
2  msp        all 22.43s   73.47%  6.42%  1.641%  tot_sequProg_busyTime               -
3  msp        1   22.43s   73.47%  6.42%  1.641%  instance_sequProg_busyTime          PSU
4  msp        all 7.35s    24.07%  2.10%  0.538%  tot_asyncIO_busyTime                -
5  msp        1   7.35s    24.07%  2.10%  0.538%  instance_asyncIO_busyTime           PSU
6  msp        all 29.78s   97.54%  8.52%  2.179%  tot_process_busyTime                -
7  msp        1   29.78s   97.54%  8.52%  2.179%  instance_busyTime                   PSU
7a msp        1   30.53s   100.00% 8.73%  2.234%  tot_busyTime(corrected)             PSU
8  msp        all 30.53s   100.00% 8.73%  2.234%  tot_busyTime(system)                -

1  oft        all 1368.10s -       -      -       duration
2  oft        all 17.69s   74.67%  5.06%  1.294%  tot_sequProg_busyTime               -
3  oft        1   17.69s   74.67%  5.06%  1.294%  instance_sequProg_busyTime          FCU
4  oft        all 5.48s    23.13%  1.57%  0.401%  tot_asyncIO_busyTime                -
```

```
5  oft        1   5.48s    23.13%  1.57%  0.401%  instance_asyncIO_busyTime            FCU
6  oft        all 23.17s   97.80%  6.63%  1.695%  tot_process_busyTime                 -
7  oft        1   23.17s   97.80%  6.63%  1.695%  instance_busyTime                    FCU
7a oft        1   23.69s   100.00% 6.78%  1.733%  tot_busyTime(corrected)              FCU
8  oft        all 23.69s   100.00% 6.78%  1.733%  tot_busyTime(system)                 -

1  opcexe     all 1369.18s -       -      -       duration
2  opcexe     all 0.58s    42.34%  0.17%  0.042%  tot_sequProg_busyTime                -
3  opcexe     1   0.58s    42.34%  0.17%  0.042%  instance_sequProg_busyTime           FCU
4  opcexe     all 0.70s    51.09%  0.20%  0.051%  tot_asyncIO_busyTime                 -
5  opcexe     1   0.70s    51.09%  0.20%  0.051%  instance_asyncIO_busyTime            FCU
6  opcexe     all 1.28s    93.43%  0.37%  0.094%  tot_process_busyTime                 -
7  opcexe     1   1.28s    93.43%  0.37%  0.094%  instance_busyTime                    FCU
7a opcexe     1   1.37s    100.00% 0.39%  0.100%  tot_busyTime(corrected)              FCU
8  opcexe     all 1.37s    100.00% 0.39%  0.100%  tot_busyTime(system)                 -

1  opcsys     all 1370.86s -       -      -       duration
2  opcsys     all 0.45s    44.12%  0.13%  0.033%  tot_sequProg_busyTime                -
3  opcsys     1   0.45s    44.12%  0.13%  0.033%  instance_sequProg_busyTime           FCU
4  opcsys     all 0.53s    51.96%  0.15%  0.039%  tot_asyncIO_busyTime                 -
5  opcsys     1   0.53s    51.96%  0.15%  0.039%  instance_asyncIO_busyTime            FCU
6  opcsys     all 0.98s    96.08%  0.28%  0.072%  tot_process_busyTime                 -
7  opcsys     1   0.98s    96.08%  0.28%  0.072%  instance_busyTime                    FCU
7a opcsys     1   1.02s    100.00% 0.29%  0.075%  tot_busyTime(corrected)              FCU
8  opcsys     all 1.02s    100.00% 0.29%  0.075%  tot_busyTime(system)                 -

1  pcs        all 1376.18s -       -      -       duration
2  pcs        all 0.15s    55.56%  0.04%  0.011%  tot_sequProg_busyTime                -
3  pcs        1   0.15s    55.56%  0.04%  0.011%  instance_sequProg_busyTime           PSU
4  pcs        all 0.05s    18.52%  0.01%  0.004%  tot_asyncIO_busyTime                 -
5  pcs        1   0.05s    18.52%  0.01%  0.004%  instance_asyncIO_busyTime            PSU
6  pcs        all 0.20s    74.07%  0.06%  0.015%  tot_process_busyTime                 -
7  pcs        1   0.20s    74.07%  0.06%  0.015%  instance_busyTime                    PSU
7a pcs        1   0.27s    100.00% 0.08%  0.020%  tot_busyTime(corrected)              PSU
8  pcs        all 0.27s    100.00% 0.08%  0.020%  tot_busyTime(system)                 -

1  qde        all 1376.81s -       -      -       duration
2  qde        all 0.21s    48.84%  0.06%  0.015%  tot_sequProg_busyTime                -
3  qde        1   0.21s    48.84%  0.06%  0.015%  instance_sequProg_busyTime           PSU
```

```
4  qde        all 0.14s    32.56%  0.04%  0.010%  tot_asyncIO_busyTime                -
5  qde        1   0.14s    32.56%  0.04%  0.010%  instance_asyncIO_busyTime           PSU
6  qde        all 0.35s    81.40%  0.10%  0.026%  tot_process_busyTime                -
7  qde        1   0.35s    81.40%  0.10%  0.026%  instance_busyTime                   PSU
7a qde        1   0.43s    100.00% 0.12%  0.031%  tot_busyTime(corrected)             PSU
8  qde        all 0.43s    100.00% 0.12%  0.031%  tot_busyTime(system)                -

1  rgv        all 1376.99s -       -      -       duration
2  rgv        all 0.13s    54.17%  0.04%  0.010%  tot_sequProg_busyTime               -
3  rgv        1   0.13s    54.17%  0.04%  0.010%  instance_sequProg_busyTime          FCU
4  rgv        all 0.08s    33.33%  0.02%  0.006%  tot_asyncIO_busyTime                -
5  rgv        1   0.08s    33.33%  0.02%  0.006%  instance_asyncIO_busyTime           FCU
6  rgv        all 0.21s    87.50%  0.06%  0.015%  tot_process_busyTime                -
7  rgv        1   0.21s    87.50%  0.06%  0.015%  instance_busyTime                   FCU
7a rgv        1   0.24s    100.00% 0.07%  0.018%  tot_busyTime(corrected)             FCU
8  rgv        all 0.24s    100.00% 0.07%  0.018%  tot_busyTime(system)                -

1  scf        all 1370.22s -       -      -       duration
2  scf        all 0.51s    51.00%  0.15%  0.037%  tot_sequProg_busyTime               -
3  scf        1   0.51s    51.00%  0.15%  0.037%  instance_sequProg_busyTime          FCU
4  scf        all 0.41s    41.00%  0.12%  0.030%  tot_asyncIO_busyTime                -
5  scf        1   0.41s    41.00%  0.12%  0.030%  instance_asyncIO_busyTime           FCU
6  scf        all 0.92s    92.00%  0.26%  0.067%  tot_process_busyTime                -
7  scf        1   0.92s    92.00%  0.26%  0.067%  instance_busyTime                   FCU
7a scf        1   1.00s    100.00% 0.29%  0.073%  tot_busyTime(corrected)             FCU
8  scf        all 1.00s    100.00% 0.29%  0.073%  tot_busyTime(system)                -

1  scr        all 1370.88s -       -      -       duration
2  scr        all 0.10s    41.67%  0.03%  0.007%  tot_sequProg_busyTime               -
3  scr        1   0.10s    41.67%  0.03%  0.007%  instance_sequProg_busyTime          FCU
4  scr        all 0.06s    25.00%  0.02%  0.004%  tot_asyncIO_busyTime                -
5  scr        1   0.06s    25.00%  0.02%  0.004%  instance_asyncIO_busyTime           FCU
6  scr        all 0.16s    66.67%  0.05%  0.012%  tot_process_busyTime                -
7  scr        1   0.16s    66.67%  0.05%  0.012%  instance_busyTime                   FCU
7a scr        1   0.24s    100.00% 0.07%  0.018%  tot_busyTime(corrected)             FCU
8  scr        all 0.24s    100.00% 0.07%  0.018%  tot_busyTime(system)                -

1  sdmsp      all 1366.84s -       -      -       duration
2  sdmsp      all 11.46s   58.92%  3.28%  0.838%  tot_sequProg_busyTime               -
```

```
3  sdmsp      1   11.46s   58.92%  3.28%  0.838%  instance_sequProg_busyTime            PSU
4  sdmsp      all 7.68s    39.49%  2.20%  0.562%  tot_asyncIO_busyTime                  -
5  sdmsp      1   7.68s    39.49%  2.20%  0.562%  instance_asyncIO_busyTime             PSU
6  sdmsp      all 19.14s   98.41%  5.47%  1.400%  tot_process_busyTime                  -
7  sdmsp      1   19.14s   98.41%  5.47%  1.400%  instance_busyTime                     PSU
7a sdmsp      1   19.45s   100.00% 5.56%  1.423%  tot_busyTime(corrected)               PSU
8  sdmsp      all 19.45s   100.00% 5.56%  1.423%  tot_busyTime(system)                  -

1  sdpyr      all 1373.43s -       -      -       duration
2  sdpyr      all 4.59s    46.98%  1.31%  0.336%  tot_sequProg_busyTime                 -
3  sdpyr      1   4.59s    46.98%  1.31%  0.336%  instance_sequProg_busyTime            FCU
4  sdpyr      all 4.96s    50.77%  1.42%  0.363%  tot_asyncIO_busyTime                  -
5  sdpyr      1   4.96s    50.77%  1.42%  0.363%  instance_asyncIO_busyTime             FCU
6  sdpyr      all 9.55s    97.75%  2.73%  0.699%  tot_process_busyTime                  -
7  sdpyr      1   9.55s    97.75%  2.73%  0.699%  instance_busyTime                     FCU
7a sdpyr      1   9.77s    100.00% 2.79%  0.715%  tot_busyTime(corrected)               FCU
8  sdpyr      all 9.77s    100.00% 2.79%  0.715%  tot_busyTime(system)                  -

1  sdt1       all 1373.76s -       -      -       duration
2  sdt1       all 27.97s   76.50%  8.00%  2.046%  tot_sequProg_busyTime                 -
3  sdt1       1   27.97s   76.50%  8.00%  2.046%  instance_sequProg_busyTime            FCU
3  sdt1       2   0.00s    0.00%   0.00%  0.000%  instance_sequProg_busyTime            FCU
3  sdt1       3   0.00s    0.00%   0.00%  0.000%  instance_sequProg_busyTime            FCU
3  sdt1       4   0.00s    0.00%   0.00%  0.000%  instance_sequProg_busyTime            PSU
4  sdt1       all 7.96s    21.77%  2.28%  0.582%  tot_asyncIO_busyTime                  -
5  sdt1       1   3.67s    10.04%  1.05%  0.269%  instance_asyncIO_busyTime             FCU
5  sdt1       2   1.31s    3.58%   0.37%  0.096%  instance_asyncIO_busyTime             FCU
5  sdt1       3   1.55s    4.24%   0.44%  0.113%  instance_asyncIO_busyTime             FCU
5  sdt1       4   1.43s    3.91%   0.41%  0.105%  instance_asyncIO_busyTime             PSU
6  sdt1       all 35.93s   98.28%  10.28% 2.629%  tot_process_busyTime                  -
7  sdt1       1   31.64s   86.54%  9.05%  2.315%  instance_busyTime                     FCU
7a sdt1       1   32.19s   88.06%  9.21%  2.355%  tot_busyTime(corrected)               FCU
7  sdt1       2   1.31s    3.58%   0.37%  0.096%  instance_busyTime                     FCU
7a sdt1       2   1.33s    3.65%   0.38%  0.098%  tot_busyTime(corrected)               FCU
7  sdt1       3   1.55s    4.24%   0.44%  0.113%  instance_busyTime                     FCU
7a sdt1       3   1.58s    4.31%   0.45%  0.115%  tot_busyTime(corrected)               FCU
7  sdt1       4   1.43s    3.91%   0.41%  0.105%  instance_busyTime                     PSU
7a sdt1       4   1.46s    3.98%   0.42%  0.106%  tot_busyTime(corrected)               PSU
8  sdt1       all 36.56s   100.00% 10.46% 2.675%  tot_busyTime(system)                  -
```

```
1  sdtmp      all 1370.74s -       -      -       duration
2  sdtmp      all 12.38s   58.95%  3.54%  0.906%  tot_sequProg_busyTime                -
3  sdtmp      1   12.38s   58.95%  3.54%  0.906%  instance_sequProg_busyTime           PSU
3  sdtmp      2   0.00s    0.00%   0.00%  0.000%  instance_sequProg_busyTime           PSU
4  sdtmp      all 8.12s    38.67%  2.32%  0.594%  tot_asyncIO_busyTime                 -
5  sdtmp      1   4.42s    21.05%  1.26%  0.323%  instance_asyncIO_busyTime            PSU
5  sdtmp      2   3.70s    17.62%  1.06%  0.271%  instance_asyncIO_busyTime            PSU
6  sdtmp      all 20.50s   97.62%  5.86%  1.500%  tot_process_busyTime                 -
7  sdtmp      1   16.80s   80.00%  4.81%  1.229%  instance_busyTime                    PSU
7a sdtmp      1   17.21s   81.95%  4.92%  1.259%  tot_busyTime(corrected)              PSU
7  sdtmp      2   3.70s    17.62%  1.06%  0.271%  instance_busyTime                    PSU
7a sdtmp      2   3.79s    18.05%  1.08%  0.277%  tot_busyTime(corrected)              PSU
8  sdtmp      all 21.00s   100.00% 6.01%  1.536%  tot_busyTime(system)                 -

1  sysinit    all 1371.02s -       -      -       duration
2  sysinit    all 0.62s    50.00%  0.18%  0.045%  tot_sequProg_busyTime                -
3  sysinit    1   0.62s    50.00%  0.18%  0.045%  instance_sequProg_busyTime           FCU
4  sysinit    all 0.12s    9.68%   0.03%  0.009%  tot_asyncIO_busyTime                 -
5  sysinit    1   0.12s    9.68%   0.03%  0.009%  instance_asyncIO_busyTime            FCU
6  sysinit    all 0.74s    59.68%  0.21%  0.054%  tot_process_busyTime                 -
7  sysinit    1   0.74s    59.68%  0.21%  0.054%  instance_busyTime                    FCU
7a sysinit    1   1.24s    100.00% 0.35%  0.091%  tot_busyTime(corrected)              FCU
8  sysinit    all 1.24s    100.00% 0.35%  0.091%  tot_busyTime(system)                 -

1  tmp        all 1374.34s -       -      -       duration
2  tmp        all 32.32s   76.35%  9.24%  2.365%  tot_sequProg_busyTime                -
3  tmp        1   32.32s   76.35%  9.24%  2.365%  instance_sequProg_busyTime           PSU
3  tmp        2   0.00s    0.00%   0.00%  0.000%  instance_sequProg_busyTime           PSU
4  tmp        all 8.73s    20.62%  2.50%  0.639%  tot_asyncIO_busyTime                 -
5  tmp        1   4.94s    11.67%  1.41%  0.361%  instance_asyncIO_busyTime            PSU
5  tmp        2   3.79s    8.95%   1.08%  0.277%  instance_asyncIO_busyTime            PSU
6  tmp        all 41.05s   96.98%  11.74% 3.003%  tot_process_busyTime                 -
7  tmp        1   37.26s   88.02%  10.66% 2.726%  instance_busyTime                    PSU
7a tmp        1   38.42s   90.77%  10.99% 2.811%  tot_busyTime(corrected)              PSU
7  tmp        2   3.79s    8.95%   1.08%  0.277%  instance_busyTime                    PSU
7a tmp        2   3.91s    9.23%   1.12%  0.286%  tot_busyTime(corrected)              PSU
8  tmp        all 42.33s   100.00% 12.11% 3.097%  tot_busyTime(system)                 -
```

```
1   tmy_cychdl all 1371.73s -       -       -       duration
2   tmy_cychdl all 0.50s    53.76%  0.14%   0.037%  tot_sequProg_busyTime               -
3   tmy_cychdl 1   0.50s    53.76%  0.14%   0.037%  instance_sequProg_busyTime          FCU
4   tmy_cychdl all 0.35s    37.63%  0.10%   0.026%  tot_asyncIO_busyTime                -
5   tmy_cychdl 1   0.35s    37.63%  0.10%   0.026%  instance_asyncIO_busyTime           FCU
6   tmy_cychdl all 0.85s    91.40%  0.24%   0.062%  tot_process_busyTime                -
7   tmy_cychdl 1   0.85s    91.40%  0.24%   0.062%  instance_busyTime                   FCU
7a  tmy_cychdl 1   0.93s    100.00% 0.27%   0.068%  tot_busyTime(corrected)             FCU
8   tmy_cychdl all 0.93s    100.00% 0.27%   0.068%  tot_busyTime(system)                -

1   tmy_reqhdl all 1374.11s -       -       -       duration
2   tmy_reqhdl all 0.33s    40.74%  0.09%   0.024%  tot_sequProg_busyTime               -
3   tmy_reqhdl 1   0.33s    40.74%  0.09%   0.024%  instance_sequProg_busyTime          FCU
4   tmy_reqhdl all 0.42s    51.85%  0.12%   0.031%  tot_asyncIO_busyTime                -
5   tmy_reqhdl 1   0.42s    51.85%  0.12%   0.031%  instance_asyncIO_busyTime           FCU
6   tmy_reqhdl all 0.75s    92.59%  0.21%   0.055%  tot_process_busyTime                -
7   tmy_reqhdl 1   0.75s    92.59%  0.21%   0.055%  instance_busyTime                   FCU
7a  tmy_reqhdl 1   0.81s    100.00% 0.23%   0.059%  tot_busyTime(corrected)             FCU
8   tmy_reqhdl all 0.81s    100.00% 0.23%   0.059%  tot_busyTime(system)                -

1   usd        all 1374.52s -       -       -       duration
2   usd        all 11.09s   77.77%  3.17%   0.811%  tot_sequProg_busyTime               -
3   usd        1   11.09s   77.77%  3.17%   0.811%  instance_sequProg_busyTime          FCU
4   usd        all 2.93s    20.55%  0.84%   0.214%  tot_asyncIO_busyTime                -
5   usd        1   2.93s    20.55%  0.84%   0.214%  instance_asyncIO_busyTime           FCU
6   usd        all 14.02s   98.32%  4.01%   1.026%  tot_process_busyTime                -
7   usd        1   14.02s   98.32%  4.01%   1.026%  instance_busyTime                   FCU
7a  usd        1   14.26s   100.00% 4.08%   1.043%  tot_busyTime(corrected)             FCU
8   usd        all 14.26s   100.00% 4.08%   1.043%  tot_busyTime(system)                -

1   vgs        all 1369.24s -       -       -       duration
2   vgs        all 0.21s    53.85%  0.06%   0.015%  tot_sequProg_busyTime               -
3   vgs        1   0.21s    53.85%  0.06%   0.015%  instance_sequProg_busyTime          FCU
4   vgs        all 0.11s    28.21%  0.03%   0.008%  tot_asyncIO_busyTime                -
5   vgs        1   0.11s    28.21%  0.03%   0.008%  instance_asyncIO_busyTime           FCU
6   vgs        all 0.32s    82.05%  0.09%   0.023%  tot_process_busyTime                -
7   vgs        1   0.32s    82.05%  0.09%   0.023%  instance_busyTime                   FCU
7a  vgs        1   0.39s    100.00% 0.11%   0.029%  tot_busyTime(corrected)             FCU
8   vgs        all 0.39s    100.00% 0.11%   0.029%  tot_busyTime(system)                -
```

```
1  vgs_sv     all 1374.18s -       -      -       duration
2  vgs_sv     all 0.11s    68.75%  0.03%  0.008%  tot_sequProg_busyTime                -
3  vgs_sv     1   0.11s    68.75%  0.03%  0.008%  instance_sequProg_busyTime           FCU
4  vgs_sv     all 0.02s    12.50%  0.01%  0.001%  tot_asyncIO_busyTime                 -
5  vgs_sv     1   0.02s    12.50%  0.01%  0.001%  instance_asyncIO_busyTime            FCU
6  vgs_sv     all 0.13s    81.25%  0.04%  0.010%  tot_process_busyTime                 -
7  vgs_sv     1   0.13s    81.25%  0.04%  0.010%  instance_busyTime                    FCU
7a vgs_sv     1   0.16s    100.00% 0.05%  0.012%  tot_busyTime(corrected)              FCU
8  vgs_sv     all 0.16s    100.00% 0.05%  0.012%  tot_busyTime(system)                 -

1  wpa        all 1377.07s -       -      -       duration
2  wpa        all 0.33s    47.14%  0.09%  0.024%  tot_sequProg_busyTime                -
3  wpa        1   0.33s    47.14%  0.09%  0.024%  instance_sequProg_busyTime           FCU
4  wpa        all 0.27s    38.57%  0.08%  0.020%  tot_asyncIO_busyTime                 -
5  wpa        1   0.27s    38.57%  0.08%  0.020%  instance_asyncIO_busyTime            FCU
6  wpa        all 0.60s    85.71%  0.17%  0.044%  tot_process_busyTime                 -
7  wpa        1   0.60s    85.71%  0.17%  0.044%  instance_busyTime                    FCU
7a wpa        1   0.70s    100.00% 0.20%  0.051%  tot_busyTime(corrected)              FCU
8  wpa        all 0.70s    100.00% 0.20%  0.051%  tot_busyTime(system)                 -

1  wpp        all 1372.77s -       -      -       duration
2  wpp        all 9.52s    73.91%  2.72%  0.696%  tot_sequProg_busyTime                -
3  wpp        1   9.52s    73.91%  2.72%  0.696%  instance_sequProg_busyTime           FCU
4  wpp        all 3.09s    23.99%  0.88%  0.226%  tot_asyncIO_busyTime                 -
5  wpp        1   3.09s    23.99%  0.88%  0.226%  instance_asyncIO_busyTime            FCU
6  wpp        all 12.61s   97.90%  3.61%  0.923%  tot_process_busyTime                 -
7  wpp        1   12.61s   97.90%  3.61%  0.923%  instance_busyTime                    FCU
7a wpp        1   12.88s   100.00% 3.68%  0.942%  tot_busyTime(corrected)              FCU
8  wpp        all 12.88s   100.00% 3.68%  0.942%  tot_busyTime(system)                 -

-  -          -   -        -       -      25.581% totalSystemUtilisation_mode=         ONETARGET

-  -          -   -        -       -      15.613% systemUtilisation_(estimated)_for_node FCU

-  -          -   -        -       -      9.973%  systemUtilisation_(estimated)_for_node PSU
```

### 7.7.2 Network Utilisation Report

The network utilisation report informs about the traffic on each CPU and between the CPU's.

When a process sends a command it indicates which (physical) channel is to be used like "udp1" below[6].Then the figures are calculated and charged to each such channel.

Again, as the information of allocation of instances to CPU's is included in the command procedure table it is possible to identify the traffic on each CPU and between the CPU's.

Each process generates such figures and they are collected by the reporting tool and and put into groups of the occured combinations of CPU's. In our case these are: FCU-FCU, PSU-PSU and FCU-PSU / PSU-FCU.

A user may specifiy three different amounts of cycles for such a physical channel which is related to the following options:

- internal traffic on a CPU

  There is an option by which a user can specifiy that UDP or Message Queues are used for internal traffic. Hence, two figures need to be provided.

- external traffic

  A third figure is needed for external communmication for each partner CPU.

Such figures are defined in file easysystem.def.

Accordingly, the following report gives the name of the used physical channel by col. 2, the involved CPU's by cols. 3 and 4, the sum of the charged cycles (col. 5), the sum of the data length (col. 6), and the sum of the product "cycles * data length" (total of consumed cycles). Finally, the last column gives the percentage of network utilisation, i.e. the total of consumed cycles times the duration of a cycle divided by the duration of the run.

Two additional figures are provided at the end: the total sum of network utilisation in case (1) that seperate channels are used for forward and backward directions, and (2) that the same channel is used for forward and backward directions.

It can clearly be seen that the network utilisation is not critical.

```
+-------------------------------------------------------
+           Network Utilisation Report
+
+-------------------------------------------------------

#1    udp1   psu fcu 1        154        154        0.00%
#2    udp1   psu fcu 1        149        149        0.00%
total udp1   psu fcu 2.00     303.00     303.00     0.00%

#1    anydev fcu fcu 893      146491     146491     0.01%
#2    anydev fcu fcu 64       10586      10586      0.00%
#3    anydev fcu fcu 4322     512156     512156     0.04%
#4    anydev fcu fcu 562      91775      91775      0.01%
```

[6] The channel "anydev" is sometimes used instead of an explicit name to leave it open which of the available channels is used.

```
#5    anydev fcu fcu 2121     229734     229734     0.02%
#6    anydev fcu fcu 2        246        246        0.00%
#7    anydev fcu fcu 1325     121907     121907     0.01%
#8    anydev fcu fcu 57       8299       8299       0.00%
#9    anydev fcu fcu 225      20778      20778      0.00%
#10   anydev fcu fcu 2        213        213        0.00%
#11   anydev fcu fcu 2757     296461     296461     0.02%
#12   anydev fcu fcu 120      19814      19814      0.00%
#13   anydev fcu fcu 80       13258      13258      0.00%
#14   anydev fcu fcu 12       1890       1890       0.00%
#15   anydev fcu fcu 71       11665      11665      0.00%
#16   anydev fcu fcu 8        1225       1225       0.00%
#17   anydev fcu fcu 905      148574     148574     0.01%
#18   anydev fcu fcu 2042     334865     334865     0.02%
#19   anydev fcu fcu 94       12886      12886      0.00%
#20   anydev fcu fcu 3        341        341        0.00%
#21   anydev fcu fcu 76       12605      12605      0.00%
#22   anydev fcu fcu 2070     222728     222728     0.02%
#23   anydev fcu fcu 25       4079       4079       0.00%
#24   anydev fcu fcu 2        213        213        0.00%
#25   anydev fcu fcu 45       7459       7459       0.00%
#26   anydev fcu fcu 2075     223887     223887     0.02%
total anydev fcu fcu 19958.00 2454135.00 2454135.00 0.18%

#1    anydev psu psu 890      146932     146932     0.01%
#2    anydev psu psu 2        192        192        0.00%
#3    anydev psu psu 2093     223404     223404     0.02%
#4    anydev psu psu 4261     451979     451979     0.03%
#5    anydev psu psu 1478     242266     242266     0.02%
#6    anydev psu psu 697      114462     114462     0.01%
#7    anydev psu psu 1368     224072     224072     0.02%
#8    anydev psu psu 4106     438400     438400     0.03%
total anydev psu psu 14895.00 1841707.00 1841707.00 0.14%

#1    anydev fcu psu 457      42044      42044      0.00%
#2    anydev fcu psu 11       1861       1861       0.00%
total anydev fcu psu 468.00   43905.00   43905.00   0.00%

#1    anydev psu fcu 95       15775      15775      0.00%
#2    anydev psu fcu 110      18311      18311      0.00%
#3    anydev psu fcu 56       9048       9048       0.00%
#4    anydev psu fcu 35       5790       5790       0.00%
#5    anydev psu fcu 498      81665      81665      0.01%
#6    anydev psu fcu 12       1981       1981       0.00%
#7    anydev psu fcu 19       3152       3152       0.00%
#8    anydev psu fcu 62       9984       9984       0.00%
total anydev psu fcu 887.00   145706.00  145706.00  0.01%
```

```
.....................................................
+          Network Summary (separated by IO direction)     +
.....................................................

#1      udp1   psu fcu 2.00     303.00     303.00     0.00%
#2      anydev psu fcu 887.00   145706.00  145706.00  0.01%
overall anydev psu fcu 889.00   146009.00  146009.00  0.01%


#1      anydev fcu fcu 19958.00 2454135.00 2454135.00 0.18%
overall anydev fcu fcu 19958.00 2454135.00 2454135.00 0.18%


#1      anydev psu psu 14895.00 1841707.00 1841707.00 0.14%
overall anydev psu psu 14895.00 1841707.00 1841707.00 0.14%


#1      anydev fcu psu 468.00   43905.00   43905.00   0.00%
overall anydev fcu psu 468.00   43905.00   43905.00   0.00%

.....................................................
+          Network Summary (merged IO directions)           +
.....................................................

#1      udp1   psu fcu 2.00     303.00     303.00     0.00%
#2      anydev fcu psu 468.00   43905.00   43905.00   0.00%
#3      anydev psu fcu 887.00   145706.00  145706.00  0.01%
overall anydev psu fcu 1357.00  189914.00  189914.00  0.01%

#1      anydev fcu fcu 19958.00 2454135.00 2454135.00 0.18%
overall anydev fcu fcu 19958.00 2454135.00 2454135.00 0.18%

#1      anydev psu psu 14895.00 1841707.00 1841707.00 0.14%
overall anydev psu psu 14895.00 1841707.00 1841707.00 0.14%
```

### 7.7.3 Timer Report

The timer report provides with a feedback on the central timer module (RSIM). This module manages all the timing events including timeout monitoring and triggering of periodic activities. This report is of interest for system tuning.

The figures on the mean length of the timer queue, its maximum length and the number of timer requests and responses may give hints for performance improvements.

The mean queue length of about 17 may be interpretd that on the average about 50% of the MSL provesses asked for timer support.

```
+-----------------------------------------------------
+           Timer Report
+-----------------------------------------------------

#TimerResponse:         11401
#TimerAutoRepetitions:  10780
#TimerRequest:          26941
#TimerDelete:           7991
#TimerInsert:           18949
MaximumQueueLength:     58
#QueueSamples:          38342
MeanValueOfQueueLength: 17.34
```

### 7.7.4 CPU Load Calibration

The report on CPU load calibration shall help a user to specifiy the amount of CPU cycles consumed during execution of a command line.

There are also more files which provide a feedback. E.g. "cyclesrep.perffeedback.file" can directly be used to define the actually consumed cycles for each line. File "perfrep.perffeedback.file" includes information about time consumption associated with the command line contents.

The load calibration report provides figures on execution of a dummy loop which is executed in order to generate CPU load in real-time mode for the missing functional code. If an engineer has information on what a loop step costs he can specifiy more precisely the figures in the command line columns for time consumption and he gets a feedback on his estimation.

The execution time per dummy loop step gives the amount of CPU time consumed per loop step. Also the observed minimum and maximum values and the computed mean value for the given number of samples are provided.

The number of loop steps executed for the command lines for a sample measurement is given by the second part. Again, minimum mean and maximum values are provided.

The final third part provides the current values which the user has specified. "currentCalibFacPerCycle" converts the number of cycles into a number of loop steps. "currentBasicCycle" is the basic time unit, usually 1μs. "#cyclesPerDummyLoopStep" gives the number of cycles represented by one loop step.

Hence, the time consumption is:

<mean execution time per dummy loop step> *

<time consumption in CPU cycles as defined by the command line> *

<currentCalibFacPerCycle> /

<#cyclesPerDummyLoopStep>

```
+-----------------------------------------------------
+           CPU Load Calibration Report
+-----------------------------------------------------

#loadCalibSamples:            22538
minExecTimePerDummyLoopStep:  0.110us
meanExecTimePerDummyLoopStep: 0.133us
maxExecTimePerDummyLoopStep:  40.350us

loopStepsMin:                 100.000
loopStepsMean:                124.408
loopStepsMax:                 199.000

User-defined data:
currentCalibFacPerCycle:      1.000
currentBasicCycle:            1.000us
#cyclesPerDummyLoopStep:      1.000
```

### 7.7.5 Response Time Report

The response time report provides information on response times and processing times. It may be used for performance monitoring or for definition of realistic timeout conditions.

To allow for measurement of response times two fields are foreseen in the standard message header: the start time and the actual time. Hence, when receiving a message the receiver can calculate the time difference against the two values and can make conclusions about the duration.

Usualy, a process which requests information from another process sets the starttime field to the actual time. The receiver sets the acttime-field to the actual time after he has processed the request. Now, the initiator of the request can measure the total response time by comparing the startime-value with the actual time and the transmission time by comparing the acttime-field with the actual time.

As the system does not know who is sender and receiver this can be defined by the command line. There is a column for the broadcasting mode of a command which may be "b" for "broadcast" (i.e. to all processes connected to a physical channel) or "s" for "selective" distribution of the command (i.e. to the process only which is defined by the destination field). If this letter is a capital letter ("B" or "S") then the starttime field is set to the actual time value automatically, hence defining the actual process as sender.

As the starttime field is only changed (if not inadvertently changed) in case a command line contains a capital letter in the broadcasting column, the starttime value can be used as a timestamp to find related events in MSC's or timing diagrams.

Response times are given for each command line whether this is meaningful or not by evaluating the difference between starttime, acttime and the actual time. For both figures the minimum, mean and maximum values are provided. the first set corresponds to the difference of actual time and starttime, the second one to actual time - acttime.

The most important information of the comamnd line, the FSM and the incoming and outgoing commands are also printed so that a user knows which action is correlated with the figures.

Only, a portion of the response time report is given.

```
+-------------------------------------------+
|           Response Time Report            |
+-------------------------------------------+
```

```
line #4 sysinit(all)/state0/subinitreturn    -> resetTimer/sameState/sysinit
          #samples=1    resp(S->D->S)=(0.000000 16.113043 16.113043)ms   resp(D->S)=(0.000000 5.867004   5.867004)ms
line #5 sysinit(all)/state0/subinitreturn    -> sub_initcmd/anyUserState/digdae
          #samples=1    resp(S->D->S)=(0.000000 30.058026 30.058026)ms   resp(D->S)=(0.000000 19.811988 19.811988)ms
line #6 sysinit(all)/state0/subinitreturn    -> timeout/anyState/sysinit
          #samples=1    resp(S->D->S)=(0.000000 30.790091 30.790091)ms   resp(D->S)=(0.000000 20.544052 20.544052)ms
line #7 sysinit(all)/state0/subinitreturn    -> noCmd/state1/sysinit
          #samples=1    resp(S->D->S)=(0.000000 31.112075 31.112075)ms   resp(D->S)=(0.000000 20.866036 20.866036)ms
line #8 sysinit(all)/state1/subinitreturn    -> resetTimer/sameState/sysinit
          #samples=1    resp(S->D->S)=(0.000000  2.915978  2.915978)ms   resp(D->S)=(0.000000  1.580000  1.580000)ms
line #9 sysinit(all)/state1/subinitreturn    -> sub_initcmd/anyUserState/anadae
          #samples=1    resp(S->D->S)=(0.000000  3.286004  3.286004)ms   resp(D->S)=(0.000000  1.950026  1.950026)ms
line #10  sysinit(all)/state1/subinitreturn  -> timeout/anyState/sysinit
          #samples=1    resp(S->D->S)=(0.000000  3.960013  3.960013)ms   resp(D->S)=(0.000000  2.624035  2.624035)ms
line #11 sysinit(all)/state1/subinitreturn   -> noCmd/state2/sysinit
          #samples=1    resp(S->D->S)=(0.000000  4.251957  4.251957)ms   resp(D->S)=(0.000000  2.915978  2.915978)ms
line #12 sysinit(all)/state2/subinitreturn   -> resetTimer/sameState/sysinit
          #samples=1    resp(S->D->S)=(0.000000 40.078998 40.078998)ms   resp(D->S)=(0.000000 18.054962 18.054962)ms


line #1  sdt1(all)/init/init_serial_line      -> daemonack/anyUserState/rts
          #samples=4    resp(S->D->S)=(0.000000 44.543982 54.134011)ms   resp(D->S)=(0.000000 19.197762 25.722027)ms
line #2  sdt1(all)/init/init_serial_line      -> noCmd/operational/sdt1
          #samples=4    resp(S->D->S)=(0.000000 48.782498 59.057951)ms   resp(D->S)=(0.000000 23.436278 39.361000)ms
line #3   sdt1(all)/operational/sdt1dae_txcmd -> hk_data/anyUserState/rts
          #samples=3367 resp(S->D->S)=(0.000000 1.670576 75.535893)ms    resp(D->S)=(0.000000  1.653837 17.751932)ms


line #1   sdt1(1)/init/init_serial_line        -> daemonack/anyUserState/rts
     #samples=1    resp(S->D->S)=(0.000000 45.418978 45.418978)ms        resp(D->S)=(0.000000 25.722027 25.722027)ms
line #2   sdt1(1)/init/init_serial_line        -> noCmd/operational/sdt1
     #samples=1    resp(S->D->S)=(0.000000 59.057951 59.057951)ms        resp(D->S)=(0.000000 39.361000 39.361000)ms
line #3   sdt1(1)/operational/sdt1dae_txcmd                  -> hk_data/anyUserState/rts
     #samples=842  resp(S->D->S)=(0.000000  1.699751 75.535893)ms        resp(D->S)=(0.000000  1.660703  6.958961)ms
```

### 7.7.6 Report on MSC Generation

The following report gives information about the number of generated MSC and timing diagram records. It is basically intended to give the tool provider information on MSC performance, e.g. how many pairs of records were found etc.

For the user it gives an idea on the CPU load which is created by generation and logging of MSC's.

```
+-----------------------------------+
+           MSC Report              |
+-----------------------------------+

MSCServer 99274 MSC records written
MSCServer 99276 timestamp records written
MSCServer: 0 erroneous records received
MSCServer: 13531 IN records
MSCServer: 13533 OUT records
MSCServer: 85745 IMM records
MSCServer: 112807 MSCview records
MSCServer: 2 remaining of queue
MSCServer: 7 maximum length of queue
MSCServer: 0 unsuccessful attempts to correlate
MSCServer: 13531 successful attempts to correlate
MSCServer: 13530 IN records BSSE/MSC
MSCServer: 13533 OUT records BSSE/MSC
MSCServer: 85744 IMM records
MSCServer: 112807 MSCview records BSSE/MSC
MSCServer: 1 remaining of queue BSSE/MSC
MSCServer: 7 maximum length of queue BSSE/MSC
MSCServer: 0 unsuccessful attempts to correlate BSSE/MSC
MSCServer: 13530 successful attempts to correlate BSSE/MSC
MSCServer: 225806 records received in total

For detailed figures of clients see file perfMSC.file
```

### 7.7.7 Command Buffer Report

Each process has an input command buffer into which incoming commands are stored[7] if they are not immediately processed in case they are member of the class "asyncstate".

When a command is requested from the command buffer it is selected according to its priority which has been assigned by the sender taking the value of the priority field of the command line.

If the figures are high (compared to the complexity of the application) either the engineer or the tool provider should think if tuning is possible.

Interesting figures are the maximum and mean queue lengths.

The maximum value may indicate an overload of the process, at least temporarily. The mean value - if high - indicates a permanent overload.

[7] In case of seperate address spaces like for UNIX-based systems (Solaris, Linux) each process has its own command buffer. If processes share the same addrss space like in case of VxWorks all processes share the same command buffer and commands are selected according to the given process id like in case of separated command buffers.

From a verification point of view it is of interest that in nearly all cases the mean value of queue length is very close to 1 or is 1. This means that no variation of buffer contents is needed as it is done in case of SDL exhaustive simulation. These figures also confirm what has been indirectly observed before when the number of system states were significantly reduced for exhaustive simulation by introduction of shared resources like CPU or channels:

> *The bottelnecks of shared resources enforce serialisation of the command transmission and release only one message after the other. The variation as performed for exhaustive simulation is equivalent to varying time delays and subsequent racings between commands.*

Hence, the variation of time consumption is more representative than combinatorial variation.

```
+-----------------------------------------+
+           Command Buffer Report         |
+-----------------------------------------+

#Samples:                   3563  for acq_hdl
#bufferStore:               1784  for acq_hdl
#bufferGet:                 3562  for acq_hdl
#bufferFree:                1781  for acq_hdl
MaximumQueueLength:         3     for acq_hdl
MeanValueOfQueueLength:     1.00  for acq_hdl
MaximumQueueLengthDev:      3     for acq_hdl
MeanValueOfQueueLengthDev: 1.00  for acq_hdl
#ExecCommand:               3563  for acq_hdl
#ExecCommandErrors:         0     for acq_hdl

#Samples:                   3567  for anadae
#bufferStore:               1783  for anadae
#bufferGet:                 3566  for anadae
#bufferFree:                1782  for anadae
MaximumQueueLength:         1     for anadae
MeanValueOfQueueLength:     1.00  for anadae
MaximumQueueLengthDev:      1     for anadae
MeanValueOfQueueLengthDev: 1.00  for anadae
#ExecCommand:               3567  for anadae
#ExecCommandErrors:         0     for anadae

#Samples:                   129   for cfv
#bufferStore:               65    for cfv
#bufferGet:                 128   for cfv
#bufferFree:                64    for cfv
MaximumQueueLength:         1     for cfv
MeanValueOfQueueLength:     1.00  for cfv
MaximumQueueLengthDev:      1     for cfv
MeanValueOfQueueLengthDev: 1.00  for cfv
#ExecCommand:               129   for cfv
#ExecCommandErrors:         0     for cfv

#Samples:                   12776 for cmdhandler
#bufferStore:               4775  for cmdhandler
#bufferGet:                 12724 for cmdhandler
#bufferFree:                4770  for cmdhandler
MaximumQueueLength:         7     for cmdhandler
MeanValueOfQueueLength:     1.91  for cmdhandler
MaximumQueueLengthDev:      7     for cmdhandler
MeanValueOfQueueLengthDev: 1.91  for cmdhandler
#ExecCommand:               12725 for cmdhandler
#ExecCommandErrors:         0     for cmdhandler

#Samples:                   1315  for digdae
```

```
#bufferStore:               658   for digdae
#bufferGet:                 1314  for digdae
#bufferFree:                657   for digdae
MaximumQueueLength:         1     for digdae
MeanValueOfQueueLength:     1.00  for digdae
MaximumQueueLengthDev:      1     for digdae
MeanValueOfQueueLengthDev: 1.00  for digdae
#ExecCommand:               1315  for digdae
#ExecCommandErrors:         0     for digdae

#Samples:                   4228  for ede
#bufferStore:               1413  for ede
#bufferGet:                 4221  for ede
#bufferFree:                1412  for ede
MaximumQueueLength:         3     for ede
MeanValueOfQueueLength:     1.02  for ede
MaximumQueueLengthDev:      3     for ede
MeanValueOfQueueLengthDev: 1.02  for ede
#ExecCommand:               4222  for ede
#ExecCommandErrors:         0     for ede

#Samples:                   1081  for emlog
#bufferStore:               537   for emlog
#bufferGet:                 1072  for emlog
#bufferFree:                536   for emlog
MaximumQueueLength:         3     for emlog
MeanValueOfQueueLength:     1.02  for emlog
MaximumQueueLengthDev:      3     for emlog
MeanValueOfQueueLengthDev: 1.02  for emlog
#ExecCommand:               1073  for emlog
#ExecCommandErrors:         0     for emlog

#Samples:                   5287  for ethdae
#bufferStore:               2643  for ethdae
#bufferGet:                 5286  for ethdae
#bufferFree:                2642  for ethdae
MaximumQueueLength:         3     for ethdae
MeanValueOfQueueLength:     1.86  for ethdae
MaximumQueueLengthDev:      3     for ethdae
MeanValueOfQueueLengthDev: 1.86  for ethdae
#ExecCommand:               5287  for ethdae
#ExecCommandErrors:         0     for ethdae

#Samples:                   1108  for fdr
#bufferStore:               529   for fdr
#bufferGet:                 1104  for fdr
#bufferFree:                528   for fdr
MaximumQueueLength:         2     for fdr
MeanValueOfQueueLength:     1.05  for fdr
MaximumQueueLengthDev:      2     for fdr
MeanValueOfQueueLengthDev: 1.05  for fdr
#ExecCommand:               1105  for fdr
#ExecCommandErrors:         0     for fdr

#Samples:                   1762  for hc_cychdl
#bufferStore:               881   for hc_cychdl
#bufferGet:                 1761  for hc_cychdl
#bufferFree:                880   for hc_cychdl
MaximumQueueLength:         1     for hc_cychdl
MeanValueOfQueueLength:     1.00  for hc_cychdl
MaximumQueueLengthDev:      1     for hc_cychdl
MeanValueOfQueueLengthDev: 1.00  for hc_cychdl
```

```
#ExecCommand:               1762  for hc_cychdl
#ExecCommandErrors:         0     for hc_cychdl

#Samples:                   223   for hc_exe
#bufferStore:               112   for hc_exe
#bufferGet:                 222   for hc_exe
#bufferFree:                111   for hc_exe
MaximumQueueLength:         1     for hc_exe
MeanValueOfQueueLength:     1.00  for hc_exe
MaximumQueueLengthDev:      1     for hc_exe
MeanValueOfQueueLengthDev: 1.00  for hc_exe
#ExecCommand:               223   for hc_exe
#ExecCommandErrors:         0     for hc_exe

#Samples:                   4267  for mfg
#bufferStore:               1420  for mfg
#bufferGet:                 4264  for mfg
#bufferFree:                1418  for mfg
MaximumQueueLength:         3     for mfg
MeanValueOfQueueLength:     1.03  for mfg
MaximumQueueLengthDev:      3     for mfg
MeanValueOfQueueLengthDev: 1.03  for mfg
#ExecCommand:               4265  for mfg
#ExecCommandErrors:         0     for mfg

#Samples:                   895   for mildae
#bufferStore:               447   for mildae
#bufferGet:                 894   for mildae
#bufferFree:                446   for mildae
MaximumQueueLength:         3     for mildae
MeanValueOfQueueLength:     1.87  for mildae
MaximumQueueLengthDev:      3     for mildae
MeanValueOfQueueLengthDev: 1.87  for mildae
#ExecCommand:               895   for mildae
#ExecCommandErrors:         0     for mildae

#Samples:                   5     for mm_dae
#bufferStore:               3     for mm_dae
#bufferGet:                 4     for mm_dae
#bufferFree:                2     for mm_dae
MaximumQueueLength:         1     for mm_dae
MeanValueOfQueueLength:     1.00  for mm_dae
MaximumQueueLengthDev:      1     for mm_dae
MeanValueOfQueueLengthDev: 1.00  for mm_dae
#ExecCommand:               5     for mm_dae
#ExecCommandErrors:         0     for mm_dae

#Samples:                   133   for ms__sv
#bufferStore:               52    for ms__sv
#bufferGet:                 132   for ms__sv
#bufferFree:                51    for ms__sv
MaximumQueueLength:         1     for ms__sv
MeanValueOfQueueLength:     1.00  for ms__sv
MaximumQueueLengthDev:      1     for ms__sv
MeanValueOfQueueLengthDev: 1.00  for ms__sv
#ExecCommand:               133   for ms__sv
#ExecCommandErrors:         0     for ms__sv

#Samples:                   9670  for msp
#bufferStore:               3287  for msp
#bufferGet:                 9639  for msp
#bufferFree:                3285  for msp
```

```
MaximumQueueLength:         8     for msp
MeanValueOfQueueLength:     1.23  for msp
MaximumQueueLengthDev:      8     for msp
MeanValueOfQueueLengthDev: 1.23  for msp
#ExecCommand:               9640  for msp
#ExecCommandErrors:         0     for msp

#Samples:                   5496  for oft
#bufferStore:               1832  for oft
#bufferGet:                 5492  for oft
#bufferFree:                1830  for oft
MaximumQueueLength:         3     for oft
MeanValueOfQueueLength:     1.01  for oft
MaximumQueueLengthDev:      3     for oft
MeanValueOfQueueLengthDev: 1.01  for oft
#ExecCommand:               5493  for oft
#ExecCommandErrors:         0     for oft

#Samples:                   241   for opcexe
#bufferStore:               121   for opcexe
#bufferGet:                 240   for opcexe
#bufferFree:                120   for opcexe
MaximumQueueLength:         1     for opcexe
MeanValueOfQueueLength:     1.00  for opcexe
MaximumQueueLengthDev:      1     for opcexe
MeanValueOfQueueLengthDev: 1.00  for opcexe
#ExecCommand:               241   for opcexe
#ExecCommandErrors:         0     for opcexe

#Samples:                   161   for opcsys
#bufferStore:               81    for opcsys
#bufferGet:                 160   for opcsys
#bufferFree:                80    for opcsys
MaximumQueueLength:         1     for opcsys
MeanValueOfQueueLength:     1.00  for opcsys
MaximumQueueLengthDev:      1     for opcsys
MeanValueOfQueueLengthDev: 1.00  for opcsys
#ExecCommand:               161   for opcsys
#ExecCommandErrors:         0     for opcsys

#Samples:                   27    for pcs
#bufferStore:               14    for pcs
#bufferGet:                 26    for pcs
#bufferFree:                13    for pcs
MaximumQueueLength:         1     for pcs
MeanValueOfQueueLength:     1.00  for pcs
MaximumQueueLengthDev:      1     for pcs
MeanValueOfQueueLengthDev: 1.00  for pcs
#ExecCommand:               27    for pcs
#ExecCommandErrors:         0     for pcs

#Samples:                   41    for qde
#bufferStore:               22    for qde
#bufferGet:                 40    for qde
#bufferFree:                20    for qde
MaximumQueueLength:         2     for qde
MeanValueOfQueueLength:     1.02  for qde
MaximumQueueLengthDev:      2     for qde
MeanValueOfQueueLengthDev: 1.02  for qde
#ExecCommand:               41    for qde
#ExecCommandErrors:         0     for qde
```

```
#Samples:                  25    for rgv
#bufferStore:              14    for rgv
#bufferGet:                24    for rgv
#bufferFree:               12    for rgv
MaximumQueueLength:        2     for rgv
MeanValueOfQueueLength:    1.04  for rgv
MaximumQueueLengthDev:     2     for rgv
MeanValueOfQueueLengthDev: 1.04  for rgv
#ExecCommand:              25    for rgv
#ExecCommandErrors:        0     for rgv

#Samples:                  151   for scf
#bufferStore:              72    for scf
#bufferGet:                150   for scf
#bufferFree:               71    for scf
MaximumQueueLength:        1     for scf
MeanValueOfQueueLength:    1.00  for scf
MaximumQueueLengthDev:     1     for scf
MeanValueOfQueueLengthDev: 1.00  for scf
#ExecCommand:              151   for scf
#ExecCommandErrors:        0     for scf

#Samples:                  17    for scr
#bufferStore:              10    for scr
#bufferGet:                16    for scr
#bufferFree:               8     for scr
MaximumQueueLength:        2     for scr
MeanValueOfQueueLength:    1.06  for scr
MaximumQueueLengthDev:     2     for scr
MeanValueOfQueueLengthDev: 1.06  for scr
#ExecCommand:              17    for scr
#ExecCommandErrors:        0     for scr

#Samples:                  2958  for sdmsp
#bufferStore:              1479  for sdmsp
#bufferGet:                2957  for sdmsp
#bufferFree:               1478  for sdmsp
MaximumQueueLength:        1     for sdmsp
MeanValueOfQueueLength:    1.00  for sdmsp
MaximumQueueLengthDev:     1     for sdmsp
MeanValueOfQueueLengthDev: 1.00  for sdmsp
#ExecCommand:              2958  for sdmsp
#ExecCommandErrors:        0     for sdmsp

#Samples:                  1812  for sdpyr
#bufferStore:              906   for sdpyr
#bufferGet:                1811  for sdpyr
#bufferFree:               905   for sdpyr
MaximumQueueLength:        1     for sdpyr
MeanValueOfQueueLength:    1.00  for sdpyr
MaximumQueueLengthDev:     1     for sdpyr
MeanValueOfQueueLengthDev: 1.00  for sdpyr
#ExecCommand:              1812  for sdpyr
#ExecCommandErrors:        0     for sdpyr

#Samples:                  5557  for sdt1
#bufferStore:              2740  for sdt1
#bufferGet:                5482  for sdt1
#bufferFree:               2739  for sdt1
MaximumQueueLength:        2     for sdt1
MeanValueOfQueueLength:    1.02  for sdt1
MaximumQueueLengthDev:     2     for sdt1
```

```
MeanValueOfQueueLengthDev: 1.02  for sdt1
#ExecCommand:              5483  for sdt1
#ExecCommandErrors:        0     for sdt1

#Samples:                  2739  for sdtmp
#bufferStore:              1369  for sdtmp
#bufferGet:                2738  for sdtmp
#bufferFree:               1368  for sdtmp
MaximumQueueLength:        1     for sdtmp
MeanValueOfQueueLength:    1.00  for sdtmp
MaximumQueueLengthDev:     1     for sdtmp
MeanValueOfQueueLengthDev: 1.00  for sdtmp
#ExecCommand:              2739  for sdtmp
#ExecCommandErrors:        0     for sdtmp

#Samples:                  179   for sysinit
#bufferStore:              38    for sysinit
#bufferGet:                178   for sysinit
#bufferFree:               37    for sysinit
MaximumQueueLength:        1     for sysinit
MeanValueOfQueueLength:    1.00  for sysinit
MaximumQueueLengthDev:     1     for sysinit
MeanValueOfQueueLengthDev: 1.00  for sysinit
#ExecCommand:              179   for sysinit
#ExecCommandErrors:        0     for sysinit

#Samples:                  9666  for tmp
#bufferStore:              2758  for tmp
#bufferGet:                9657  for tmp
#bufferFree:               2757  for tmp
MaximumQueueLength:        4     for tmp
MeanValueOfQueueLength:    1.02  for tmp
MaximumQueueLengthDev:     4     for tmp
MeanValueOfQueueLengthDev: 1.02  for tmp
#ExecCommand:              9658  for tmp
#ExecCommandErrors:        0     for tmp

#Samples:                  1326  for tmy_cychdl
#bufferStore:              664   for tmy_cychdl
#bufferGet:                1325  for tmy_cychdl
#bufferFree:               662   for tmy_cychdl
MaximumQueueLength:        2     for tmy_cychdl
MeanValueOfQueueLength:    1.00  for tmy_cychdl
MaximumQueueLengthDev:     2     for tmy_cychdl
MeanValueOfQueueLengthDev: 1.00  for tmy_cychdl
#ExecCommand:              1326  for tmy_cychdl
#ExecCommandErrors:        0     for tmy_cychdl

#Samples:                  153   for tmy_reqhdl
#bufferStore:              78    for tmy_reqhdl
#bufferGet:                152   for tmy_reqhdl
#bufferFree:               76    for tmy_reqhdl
MaximumQueueLength:        2     for tmy_reqhdl
MeanValueOfQueueLength:    1.01  for tmy_reqhdl
MaximumQueueLengthDev:     2     for tmy_reqhdl
MeanValueOfQueueLengthDev: 1.01  for tmy_reqhdl
#ExecCommand:              153   for tmy_reqhdl
#ExecCommandErrors:        0     for tmy_reqhdl

#Samples:                  4128  for usd
#bufferStore:              1377  for usd
#bufferGet:                4125  for usd
```

```
#bufferFree:                1376  for usd
MaximumQueueLength:         3     for usd
MeanValueOfQueueLength:     1.01  for usd
MaximumQueueLengthDev:      3     for usd
MeanValueOfQueueLengthDev: 1.01  for usd
#ExecCommand:               4126  for usd
#ExecCommandErrors:         0     for usd

#Samples:                   51    for vgs
#bufferStore:               27    for vgs
#bufferGet:                 50    for vgs
#bufferFree:                25    for vgs
MaximumQueueLength:         2     for vgs
MeanValueOfQueueLength:     1.02  for vgs
MaximumQueueLengthDev:      2     for vgs
MeanValueOfQueueLengthDev: 1.02  for vgs
#ExecCommand:               51    for vgs
#ExecCommandErrors:         0     for vgs

#Samples:                   5     for vgs_sv
#bufferStore:               3     for vgs_sv
#bufferGet:                 4     for vgs_sv
#bufferFree:                2     for vgs_sv
MaximumQueueLength:         1     for vgs_sv
MeanValueOfQueueLength:     1.00  for vgs_sv
MaximumQueueLengthDev:      1     for vgs_sv
MeanValueOfQueueLengthDev: 1.00  for vgs_sv
#ExecCommand:               5     for vgs_sv
#ExecCommandErrors:         0     for vgs_sv

#Samples:                   91    for wpa
#bufferStore:               46    for wpa
#bufferGet:                 90    for wpa
#bufferFree:                45    for wpa
MaximumQueueLength:         1     for wpa
MeanValueOfQueueLength:     1.00  for wpa
MaximumQueueLengthDev:      1     for wpa
MeanValueOfQueueLengthDev: 1.00  for wpa
#ExecCommand:               91    for wpa
#ExecCommandErrors:         0     for wpa

#Samples:                   4141  for wpp
#bufferStore:               1383  for wpp
#bufferGet:                 4134  for wpp
#bufferFree:                1382  for wpp
MaximumQueueLength:         2     for wpp
MeanValueOfQueueLength:     1.01  for wpp
MaximumQueueLengthDev:      2     for wpp
MeanValueOfQueueLengthDev: 1.01  for wpp
#ExecCommand:               4135  for wpp
#ExecCommandErrors:         0     for wpp
```

### 7.8 The "Stress Testing" Case

For stress testing the system processes were exposed to automated stimulation by commands except for sysinit, cmdhandler, ethdae and mildae. "sysinit" is only involved in system initialisation, cmdhandler, ethdae and mildae are already automatically stimulated in the operaional case. Therefore only the remaining processes shall be subject of stress testing.

In this mode the system issues periodically commands to the processes which leads to a creation of a speciific command of a process which is selected out of the set of commands related to its actual state.

If the destination is not explicitly defined (e.g. in case of rts or udc) the outgoing command is sent to the sender which creates error conditions and possibly cyclic loops for command processing. In case a command line is activated which starts a periodic activity the workload is heavily increased (one incoming command generates an infinite number of future events).

This way the system is exposed to stress and it is of interest to analyse if and how the system survives this attack.

#### 7.8.1 The Exception Report

The exception report has to be compared with the exception report of the operational case in order to identify locations which are sensitive for stress testing. A comparison shows that the same lines are involved and such lines are related to timeout conditions. So no new lines are identified. For occurence of wrong incoming commands the error.log-file and the MSC-file has to be analysed. A number of exceptions for incoming commands are reported.

```
+-------------------------------------------------------+
+           Exception Report                            |
+-------------------------------------------------------+

line 2  ede(all)/pre_init/daemonack      30  exceptions
line 6  ede(all)/init/hk_data            119 exceptions

line 2  mfg(all)/pre_init/daemonack      27  exceptions
line 6  mfg(all)/init/hk_data            97  exceptions

line 7  ms__sv(all)/anyState/daemonack   40  exceptions

line 2  msp(all)/pre_init/daemonack      9   exceptions
line 47 msp(all)/operational/ms_specdata 117 exceptions
line 51 msp(all)/operational/ms_hkdata   111 exceptions
line 56 msp(all)/operational/ms_trpdata  7   exceptions

line 2  oft(all)/pre_init/daemonack      29  exceptions
line 6  oft(all)/init/hk_data            103 exceptions

line 6  tmp(1)/init/hk_data              52  exceptions
line 2  tmp(2)/pre_init/daemonack        29  exceptions
line 6  tmp(2)/init/hk_data              61  exceptions
line 2  tmp(all)/pre_init/daemonack      29  exceptions
line 6  tmp(all)/init/hk_data            113 exceptions

line 2  usd(all)/pre_init/daemonack      24  exceptions
line 6  usd(all)/init/hk_data            89  exceptions

line 2  wpp(all)/pre_init/daemonack      36  exceptions
```

```
line 6  wpp(all)/init/hk_data            86  exceptions
```

### 7.8.2 The Command Buffer Report

Due to the increased load a slightly higher figure occurs for the mean queue length for most of the processes. In some exceptional cases like for fdr, hc_cychdl and tmy_cychdl the buffer is exhausted due to repeated activiation of a command line which starts a periodic timer. This can be seen by the MSC and the timing diagrams.

It has to be decided by the project if a protection is really needed against such inadvertent commands or not.

```
+-----------------------------------------------------+
+           Command Buffer Report                     |
+-----------------------------------------------------+


#Samples:                   6     for ethdae
#bufferStore:               4     for ethdae
#bufferGet:                 5     for ethdae
#bufferFree:                2     for ethdae
MaximumQueueLength:         2     for ethdae
MeanValueOfQueueLength:     1.17  for ethdae
MaximumQueueLengthDev:      2     for ethdae
MeanValueOfQueueLengthDev: 1.17  for ethdae
#ExecCommand:               6     for ethdae
#ExecCommandErrors:         0     for ethdae


#Samples:                   6     for ethdae
#bufferStore:               4     for ethdae
#bufferGet:                 5     for ethdae
#bufferFree:                2     for ethdae
MaximumQueueLength:         2     for ethdae
MeanValueOfQueueLength:     1.17  for ethdae
MaximumQueueLengthDev:      2     for ethdae
MeanValueOfQueueLengthDev: 1.17  for ethdae
#ExecCommand:               6     for ethdae
#ExecCommandErrors:         0     for ethdae

#Samples:                   3322  for fdr
#bufferStore:               1618  for fdr
#bufferGet:                 3088  for fdr
#bufferFree:                1617  for fdr
MaximumQueueLength:         200   for fdr
MeanValueOfQueueLength:     23.19 for fdr
MaximumQueueLengthDev:      199   for fdr
MeanValueOfQueueLengthDev: 23.13 for fdr
#ExecCommand:               3287  for fdr
#ExecCommandErrors:         199   for fdr

#Samples:                   2816  for hc_cychdl
#bufferStore:               1606  for hc_cychdl
#bufferGet:                 2390  for hc_cychdl
#bufferFree:                1605  for hc_cychdl
MaximumQueueLength:         200   for hc_cychdl
MeanValueOfQueueLength:     20.84 for hc_cychdl
MaximumQueueLengthDev:      199   for hc_cychdl
MeanValueOfQueueLengthDev: 20.70 for hc_cychdl
```

```
#ExecCommand:               2788  for hc_cychdl
#ExecCommandErrors:         407   for hc_cychdl

#Samples:                   530   for ms__sv
#bufferStore:               283   for ms__sv
#bufferGet:                 521   for ms__sv
#bufferFree:                280   for ms__sv
MaximumQueueLength:         4     for ms__sv
MeanValueOfQueueLength:     1.68  for ms__sv
MaximumQueueLengthDev:      4     for ms__sv
MeanValueOfQueueLengthDev: 1.68  for ms__sv
#ExecCommand:               522   for ms__sv
#ExecCommandErrors:         0     for ms__sv


#Samples:                   2549  for tmy_cychdl
#bufferStore:               1473  for tmy_cychdl
#bufferGet:                 2122  for tmy_cychdl
#bufferFree:                1472  for tmy_cychdl
MaximumQueueLength:         200   for tmy_cychdl
MeanValueOfQueueLength:     20.78 for tmy_cychdl
MaximumQueueLengthDev:      199   for tmy_cychdl
MeanValueOfQueueLengthDev: 20.62 for tmy_cychdl
#ExecCommand:               2520  for tmy_cychdl
#ExecCommandErrors:         409   for tmy_cychdl


#Samples:                   1883  for wpp
#bufferStore:               689   for wpp
#bufferGet:                 1874  for wpp
#bufferFree:                681   for wpp
MaximumQueueLength:         14    for wpp
MeanValueOfQueueLength:     1.57  for wpp
MaximumQueueLengthDev:      14    for wpp
MeanValueOfQueueLengthDev: 1.57  for wpp
#ExecCommand:               1875  for wpp
#ExecCommandErrors:         0     for wpp
```

### 7.8.3 The report on CPU Utilisation

The report on CPU utilisation shows a significantly increased figure for the CPU laod due to the automated and periodic stimulation and start of a number of cyclic activities.

However, these figures also prove that the system does tolerate such a high workload without crashing.

```
+-------------------------------------------------------+
+           Evaluation of CPU Utilisation               |
+-------------------------------------------------------+


  PROCMIN=opcsys MINDUR=597.48 PROCMAX=fdr MAXDUR=779.78


1  RSIM       all 613.08s -       -      -       duration
2  RSIM       all 0.13s   1.08%   0.03%  0.022%  tot_sequProg_busyTime                -
3  RSIM       1   0.13s   1.08%   0.03%  0.022%  instance_sequProg_busyTime
4  RSIM       all 2.66s   22.00%  0.65%  0.445%  tot_asyncIO_busyTime                 -
5  RSIM       1   2.66s   22.00%  0.65%  0.445%  instance_asyncIO_busyTime
6  RSIM       all 2.79s   23.08%  0.68%  0.467%  tot_process_busyTime                 -
7  RSIM       1   2.79s   23.08%  0.68%  0.467%  instance_busyTime
7a RSIM       1   12.09s  100.00% 2.96%  2.023%  tot_busyTime(corrected)
8  RSIM       all 12.09s  100.00% 2.96%  2.023%  tot_busyTime(system)                 -

1  acq_hdl    all 603.01s -       -      -       duration
2  acq_hdl    all 15.20s  60.41%  3.72%  2.544%  tot_sequProg_busyTime                -
3  acq_hdl    1   15.20s  60.41%  3.72%  2.544%  instance_sequProg_busyTime            FCU
3  acq_hdl    2   0.00s   0.00%   0.00%  0.000%  instance_sequProg_busyTime            PSU
4  acq_hdl    all 8.92s   35.45%  2.18%  1.493%  tot_asyncIO_busyTime                  -
5  acq_hdl    1   1.49s   5.92%   0.36%  0.249%  instance_asyncIO_busyTime             FCU
5  acq_hdl    2   7.43s   29.53%  1.82%  1.244%  instance_asyncIO_busyTime             PSU
6  acq_hdl    all 24.12s  95.87%  5.90%  4.037%  tot_process_busyTime                  -
7  acq_hdl    1   16.69s  66.34%  4.08%  2.793%  instance_busyTime                     FCU
7a acq_hdl    1   17.41s  69.20%  4.26%  2.914%  tot_busyTime(corrected)               FCU
7  acq_hdl    2   7.43s   29.53%  1.82%  1.244%  instance_busyTime                     PSU
```

```
7a acq_hdl    2   7.75s   30.80%  1.90%  1.297%  tot_busyTime(corrected)              PSU
8  acq_hdl    all 25.16s  100.00% 6.15%  4.211%  tot_busyTime(system)                 -

1  anadae     all 608.25s -       -      -       duration
2  anadae     all 33.41s  79.25%  8.17%  5.592%  tot_sequProg_busyTime                -
3  anadae     1   33.41s  79.25%  8.17%  5.592%  instance_sequProg_busyTime           FCU
3  anadae     2   0.00s   0.00%   0.00%  0.000%  instance_sequProg_busyTime           PSU
4  anadae     all 8.25s   19.57%  2.02%  1.381%  tot_asyncIO_busyTime                 -
5  anadae     1   1.13s   2.68%   0.28%  0.189%  instance_asyncIO_busyTime            FCU
5  anadae     2   7.12s   16.89%  1.74%  1.192%  instance_asyncIO_busyTime            PSU
6  anadae     all 41.66s  98.81%  10.19% 6.973%  tot_process_busyTime                 -
7  anadae     1   34.54s  81.93%  8.45%  5.781%  instance_busyTime                    FCU
7a anadae     1   34.95s  82.91%  8.55%  5.850%  tot_busyTime(corrected)              FCU
7  anadae     2   7.12s   16.89%  1.74%  1.192%  instance_busyTime                    PSU
7a anadae     2   7.21s   17.09%  1.76%  1.206%  tot_busyTime(corrected)              PSU
8  anadae     all 42.16s  100.00% 10.31% 7.056%  tot_busyTime(system)                 -

1  cfv        all 612.26s -       -      -       duration
2  cfv        all 2.32s   72.05%  0.57%  0.388%  tot_sequProg_busyTime                -
3  cfv        1   2.32s   72.05%  0.57%  0.388%  instance_sequProg_busyTime           FCU
4  cfv        all 0.80s   24.84%  0.20%  0.134%  tot_asyncIO_busyTime                 -
5  cfv        1   0.80s   24.84%  0.20%  0.134%  instance_asyncIO_busyTime            FCU
6  cfv        all 3.12s   96.89%  0.76%  0.522%  tot_process_busyTime                 -
7  cfv        1   3.12s   96.89%  0.76%  0.522%  instance_busyTime                    FCU
7a cfv        1   3.22s   100.00% 0.79%  0.539%  tot_busyTime(corrected)              FCU
8  cfv        all 3.22s   100.00% 0.79%  0.539%  tot_busyTime(system)                 -

1  cmdhandler all 611.97s -       -      -       duration
2  cmdhandler all 0.06s   40.00%  0.01%  0.010%  tot_sequProg_busyTime                -
3  cmdhandler 1   0.06s   40.00%  0.01%  0.010%  instance_sequProg_busyTime           FCU
4  cmdhandler all 0.01s   6.67%   0.00%  0.002%  tot_asyncIO_busyTime                 -
5  cmdhandler 1   0.01s   6.67%   0.00%  0.002%  instance_asyncIO_busyTime            FCU
6  cmdhandler all 0.07s   46.67%  0.02%  0.012%  tot_process_busyTime                 -
7  cmdhandler 1   0.07s   46.67%  0.02%  0.012%  instance_busyTime                    FCU
7a cmdhandler 1   0.15s   100.00% 0.04%  0.025%  tot_busyTime(corrected)              FCU
8  cmdhandler all 0.15s   100.00% 0.04%  0.025%  tot_busyTime(system)                 -

1  digdae     all 617.92s -       -      -       duration
```

```
2  digdae     all 2.27s   57.61%  0.56%  0.380%  tot_sequProg_busyTime              -
3  digdae     1   2.27s   57.61%  0.56%  0.380%  instance_sequProg_busyTime         FCU
3  digdae     2   0.00s   0.00%   0.00%  0.000%  instance_sequProg_busyTime         PSU
4  digdae     all 0.83s   21.07%  0.20%  0.139%  tot_asyncIO_busyTime               -
5  digdae     1   0.83s   21.07%  0.20%  0.139%  instance_asyncIO_busyTime          FCU
5  digdae     2   0.00s   0.00%   0.00%  0.000%  instance_asyncIO_busyTime          PSU
6  digdae     all 3.10s   78.68%  0.76%  0.519%  tot_process_busyTime               -
7  digdae     1   3.10s   78.68%  0.76%  0.519%  instance_busyTime                  FCU
7a digdae     1   3.94s   100.00% 0.96%  0.659%  tot_busyTime(corrected)            FCU
7  digdae     2   0.00s   0.00%   0.00%  0.000%  instance_busyTime                  PSU
7a digdae     2   0.00s   0.00%   0.00%  0.000%  tot_busyTime(corrected)            PSU
8  digdae     all 3.94s   100.00% 0.96%  0.659%  tot_busyTime(system)               -

1  ede        all 610.09s -       -      -       duration
2  ede        all 9.78s   78.18%  2.39%  1.637%  tot_sequProg_busyTime              -
3  ede        1   9.78s   78.18%  2.39%  1.637%  instance_sequProg_busyTime         FCU
4  ede        all 2.32s   18.55%  0.57%  0.388%  tot_asyncIO_busyTime               -
5  ede        1   2.32s   18.55%  0.57%  0.388%  instance_asyncIO_busyTime          FCU
6  ede        all 12.10s  96.72%  2.96%  2.025%  tot_process_busyTime               -
7  ede        1   12.10s  96.72%  2.96%  2.025%  instance_busyTime                  FCU
7a ede        1   12.51s  100.00% 3.06%  2.094%  tot_busyTime(corrected)            FCU
8  ede        all 12.51s  100.00% 3.06%  2.094%  tot_busyTime(system)               -

1  emlog      all 604.25s -       -      -       duration
2  emlog      all 6.76s   53.78%  1.65%  1.131%  tot_sequProg_busyTime              -
3  emlog      1   6.76s   53.78%  1.65%  1.131%  instance_sequProg_busyTime         FCU
4  emlog      all 5.52s   43.91%  1.35%  0.924%  tot_asyncIO_busyTime               -
5  emlog      1   5.52s   43.91%  1.35%  0.924%  instance_asyncIO_busyTime          FCU
6  emlog      all 12.28s  97.69%  3.00%  2.055%  tot_process_busyTime               -
7  emlog      1   12.28s  97.69%  3.00%  2.055%  instance_busyTime                  FCU
7a emlog      1   12.57s  100.00% 3.07%  2.104%  tot_busyTime(corrected)            FCU
8  emlog      all 12.57s  100.00% 3.07%  2.104%  tot_busyTime(system)               -

1  ethdae     all 602.50s -       -      -       duration
2  ethdae     all 0.06s   37.50%  0.01%  0.010%  tot_sequProg_busyTime              -
3  ethdae     1   0.06s   37.50%  0.01%  0.010%  instance_sequProg_busyTime         FCU
4  ethdae     all 0.02s   12.50%  0.00%  0.003%  tot_asyncIO_busyTime               -
5  ethdae     1   0.02s   12.50%  0.00%  0.003%  instance_asyncIO_busyTime          FCU
```

```
6  ethdae     all 0.08s   50.00%  0.02%  0.013%  tot_process_busyTime                -
7  ethdae     1   0.08s   50.00%  0.02%  0.013%  instance_busyTime                   FCU
7a ethdae     1   0.16s   100.00% 0.04%  0.027%  tot_busyTime(corrected)             FCU
8  ethdae     all 0.16s   100.00% 0.04%  0.027%  tot_busyTime(system)                -

1  fdr        all 779.78s -       -      -       duration
2  fdr        all 13.68s  58.84%  3.35%  2.290%  tot_sequProg_busyTime               -
3  fdr        1   13.68s  58.84%  3.35%  2.290%  instance_sequProg_busyTime          FCU
4  fdr        all 8.82s   37.94%  2.16%  1.476%  tot_asyncIO_busyTime                -
5  fdr        1   8.82s   37.94%  2.16%  1.476%  instance_asyncIO_busyTime           FCU
6  fdr        all 22.50s  96.77%  5.50%  3.766%  tot_process_busyTime                -
7  fdr        1   22.50s  96.77%  5.50%  3.766%  instance_busyTime                   FCU
7a fdr        1   23.25s  100.00% 5.69%  3.891%  tot_busyTime(corrected)             FCU
8  fdr        all 23.25s  100.00% 5.69%  3.891%  tot_busyTime(system)                -

1  hc_cychdl  all 778.06s -       -      -       duration
2  hc_cychdl  all 16.51s  30.74%  4.04%  2.763%  tot_sequProg_busyTime               -
3  hc_cychdl  1   16.51s  30.74%  4.04%  2.763%  instance_sequProg_busyTime          PSU
4  hc_cychdl  all 36.30s  67.59%  8.88%  6.076%  tot_asyncIO_busyTime                -
5  hc_cychdl  1   36.30s  67.59%  8.88%  6.076%  instance_asyncIO_busyTime           PSU
6  hc_cychdl  all 52.81s  98.32%  12.92% 8.839%  tot_process_busyTime                -
7  hc_cychdl  1   52.81s  98.32%  12.92% 8.839%  instance_busyTime                   PSU
7a hc_cychdl  1   53.71s  100.00% 13.14% 8.989%  tot_busyTime(corrected)             PSU
8  hc_cychdl  all 53.71s  100.00% 13.14% 8.989%  tot_busyTime(system)                -

1  hc_exe     all 610.65s -       -      -       duration
2  hc_exe     all 2.06s   75.18%  0.50%  0.345%  tot_sequProg_busyTime               -
3  hc_exe     1   2.06s   75.18%  0.50%  0.345%  instance_sequProg_busyTime          PSU
4  hc_exe     all 0.58s   21.17%  0.14%  0.097%  tot_asyncIO_busyTime                -
5  hc_exe     1   0.58s   21.17%  0.14%  0.097%  instance_asyncIO_busyTime           PSU
6  hc_exe     all 2.64s   96.35%  0.65%  0.442%  tot_process_busyTime                -
7  hc_exe     1   2.64s   96.35%  0.65%  0.442%  instance_busyTime                   PSU
7a hc_exe     1   2.74s   100.00% 0.67%  0.459%  tot_busyTime(corrected)             PSU
8  hc_exe     all 2.74s   100.00% 0.67%  0.459%  tot_busyTime(system)                -

1  mfg        all 610.01s -       -      -       duration
2  mfg        all 9.16s   76.14%  2.24%  1.533%  tot_sequProg_busyTime               -
3  mfg        1   9.16s   76.14%  2.24%  1.533%  instance_sequProg_busyTime          PSU
```

```
4  mfg        all 2.53s   21.03%  0.62%  0.423%  tot_asyncIO_busyTime              -
5  mfg        1   2.53s   21.03%  0.62%  0.423%  instance_asyncIO_busyTime         PSU
6  mfg        all 11.69s  97.17%  2.86%  1.957%  tot_process_busyTime              -
7  mfg        1   11.69s  97.17%  2.86%  1.957%  instance_busyTime                 PSU
7a mfg        1   12.03s  100.00% 2.94%  2.013%  tot_busyTime(corrected)           PSU
8  mfg        all 12.03s  100.00% 2.94%  2.013%  tot_busyTime(system)              -

1  mildae     all 611.56s -       -      -       duration
2  mildae     all 0.10s   66.67%  0.02%  0.017%  tot_sequProg_busyTime             -
3  mildae     1   0.10s   66.67%  0.02%  0.017%  instance_sequProg_busyTime        FCU
4  mildae     all 0.01s   6.67%   0.00%  0.002%  tot_asyncIO_busyTime              -
5  mildae     1   0.01s   6.67%   0.00%  0.002%  instance_asyncIO_busyTime         FCU
6  mildae     all 0.11s   73.33%  0.03%  0.018%  tot_process_busyTime              -
7  mildae     1   0.11s   73.33%  0.03%  0.018%  instance_busyTime                 FCU
7a mildae     1   0.15s   100.00% 0.04%  0.025%  tot_busyTime(corrected)           FCU
8  mildae     all 0.15s   100.00% 0.04%  0.025%  tot_busyTime(system)              -

1  mm_dae     all 603.47s -       -      -       duration
2  mm_dae     all 1.50s   77.72%  0.37%  0.251%  tot_sequProg_busyTime             -
3  mm_dae     1   1.50s   77.72%  0.37%  0.251%  instance_sequProg_busyTime        FCU
4  mm_dae     all 0.38s   19.69%  0.09%  0.064%  tot_asyncIO_busyTime              -
5  mm_dae     1   0.38s   19.69%  0.09%  0.064%  instance_asyncIO_busyTime         FCU
6  mm_dae     all 1.88s   97.41%  0.46%  0.315%  tot_process_busyTime              -
7  mm_dae     1   1.88s   97.41%  0.46%  0.315%  instance_busyTime                 FCU
7a mm_dae     1   1.93s   100.00% 0.47%  0.323%  tot_busyTime(corrected)           FCU
8  mm_dae     all 1.93s   100.00% 0.47%  0.323%  tot_busyTime(system)              -

1  ms__sv     all 605.79s -       -      -       duration
2  ms__sv     all 3.11s   78.93%  0.76%  0.521%  tot_sequProg_busyTime             -
3  ms__sv     1   3.11s   78.93%  0.76%  0.521%  instance_sequProg_busyTime        PSU
4  ms__sv     all 0.73s   18.53%  0.18%  0.122%  tot_asyncIO_busyTime              -
5  ms__sv     1   0.73s   18.53%  0.18%  0.122%  instance_asyncIO_busyTime         PSU
6  ms__sv     all 3.84s   97.46%  0.94%  0.643%  tot_process_busyTime              -
7  ms__sv     1   3.84s   97.46%  0.94%  0.643%  instance_busyTime                 PSU
7a ms__sv     1   3.94s   100.00% 0.96%  0.659%  tot_busyTime(corrected)           PSU
8  ms__sv     all 3.94s   100.00% 0.96%  0.659%  tot_busyTime(system)              -

1  msp        all 624.48s -       -      -       duration
```

```
2  msp        all 15.51s  75.58%  3.79%  2.596%  tot_sequProg_busyTime              -
3  msp        1   15.51s  75.58%  3.79%  2.596%  instance_sequProg_busyTime         PSU
4  msp        all 4.19s   20.42%  1.02%  0.701%  tot_asyncIO_busyTime               -
5  msp        1   4.19s   20.42%  1.02%  0.701%  instance_asyncIO_busyTime          PSU
6  msp        all 19.70s  96.00%  4.82%  3.297%  tot_process_busyTime               -
7  msp        1   19.70s  96.00%  4.82%  3.297%  instance_busyTime                  PSU
7a msp        1   20.52s  100.00% 5.02%  3.434%  tot_busyTime(corrected)            PSU
8  msp        all 20.52s  100.00% 5.02%  3.434%  tot_busyTime(system)               -

1  oft        all 613.88s -       -      -       duration
2  oft        all 10.75s  77.56%  2.63%  1.799%  tot_sequProg_busyTime              -
3  oft        1   10.75s  77.56%  2.63%  1.799%  instance_sequProg_busyTime         FCU
4  oft        all 2.82s   20.35%  0.69%  0.472%  tot_asyncIO_busyTime               -
5  oft        1   2.82s   20.35%  0.69%  0.472%  instance_asyncIO_busyTime          FCU
6  oft        all 13.57s  97.91%  3.32%  2.271%  tot_process_busyTime               -
7  oft        1   13.57s  97.91%  3.32%  2.271%  instance_busyTime                  FCU
7a oft        1   13.86s  100.00% 3.39%  2.320%  tot_busyTime(corrected)            FCU
8  oft        all 13.86s  100.00% 3.39%  2.320%  tot_busyTime(system)               -

1  opcexe     all 608.46s -       -      -       duration
2  opcexe     all 2.03s   71.73%  0.50%  0.340%  tot_sequProg_busyTime              -
3  opcexe     1   2.03s   71.73%  0.50%  0.340%  instance_sequProg_busyTime         FCU
4  opcexe     all 0.65s   22.97%  0.16%  0.109%  tot_asyncIO_busyTime               -
5  opcexe     1   0.65s   22.97%  0.16%  0.109%  instance_asyncIO_busyTime          FCU
6  opcexe     all 2.68s   94.70%  0.66%  0.449%  tot_process_busyTime               -
7  opcexe     1   2.68s   94.70%  0.66%  0.449%  instance_busyTime                  FCU
7a opcexe     1   2.83s   100.00% 0.69%  0.474%  tot_busyTime(corrected)            FCU
8  opcexe     all 2.83s   100.00% 0.69%  0.474%  tot_busyTime(system)               -

1  opcsys     all 597.48s -       -      -       duration
2  opcsys     all 2.09s   73.33%  0.51%  0.350%  tot_sequProg_busyTime              -
3  opcsys     1   2.09s   73.33%  0.51%  0.350%  instance_sequProg_busyTime         FCU
4  opcsys     all 0.67s   23.51%  0.16%  0.112%  tot_asyncIO_busyTime               -
5  opcsys     1   0.67s   23.51%  0.16%  0.112%  instance_asyncIO_busyTime          FCU
6  opcsys     all 2.76s   96.84%  0.68%  0.462%  tot_process_busyTime               -
7  opcsys     1   2.76s   96.84%  0.68%  0.462%  instance_busyTime                  FCU
7a opcsys     1   2.85s   100.00% 0.70%  0.477%  tot_busyTime(corrected)            FCU
8  opcsys     all 2.85s   100.00% 0.70%  0.477%  tot_busyTime(system)               -
```

```
1  pcs        all 612.45s -       -      -       duration
2  pcs        all 2.28s   77.29%  0.56%  0.382%  tot_sequProg_busyTime                -
3  pcs        1   2.28s   77.29%  0.56%  0.382%  instance_sequProg_busyTime           PSU
4  pcs        all 0.57s   19.32%  0.14%  0.095%  tot_asyncIO_busyTime                 -
5  pcs        1   0.57s   19.32%  0.14%  0.095%  instance_asyncIO_busyTime            PSU
6  pcs        all 2.85s   96.61%  0.70%  0.477%  tot_process_busyTime                 -
7  pcs        1   2.85s   96.61%  0.70%  0.477%  instance_busyTime                    PSU
7a pcs        1   2.95s   100.00% 0.72%  0.494%  tot_busyTime(corrected)              PSU
8  pcs        all 2.95s   100.00% 0.72%  0.494%  tot_busyTime(system)                 -

1  qde        all 609.64s -       -      -       duration
2  qde        all 2.23s   75.85%  0.55%  0.373%  tot_sequProg_busyTime                -
3  qde        1   2.23s   75.85%  0.55%  0.373%  instance_sequProg_busyTime           PSU
4  qde        all 0.60s   20.41%  0.15%  0.100%  tot_asyncIO_busyTime                 -
5  qde        1   0.60s   20.41%  0.15%  0.100%  instance_asyncIO_busyTime            PSU
6  qde        all 2.83s   96.26%  0.69%  0.474%  tot_process_busyTime                 -
7  qde        1   2.83s   96.26%  0.69%  0.474%  instance_busyTime                    PSU
7a qde        1   2.94s   100.00% 0.72%  0.492%  tot_busyTime(corrected)              PSU
8  qde        all 2.94s   100.00% 0.72%  0.492%  tot_busyTime(system)                 -

1  rgv        all 606.58s -       -      -       duration
2  rgv        all 2.56s   77.81%  0.63%  0.428%  tot_sequProg_busyTime                -
3  rgv        1   2.56s   77.81%  0.63%  0.428%  instance_sequProg_busyTime           FCU
4  rgv        all 0.64s   19.45%  0.16%  0.107%  tot_asyncIO_busyTime                 -
5  rgv        1   0.64s   19.45%  0.16%  0.107%  instance_asyncIO_busyTime            FCU
6  rgv        all 3.20s   97.26%  0.78%  0.536%  tot_process_busyTime                 -
7  rgv        1   3.20s   97.26%  0.78%  0.536%  instance_busyTime                    FCU
7a rgv        1   3.29s   100.00% 0.80%  0.551%  tot_busyTime(corrected)              FCU
8  rgv        all 3.29s   100.00% 0.80%  0.551%  tot_busyTime(system)                 -

1  scf        all 611.29s -       -      -       duration
2  scf        all 2.34s   76.22%  0.57%  0.392%  tot_sequProg_busyTime                -
3  scf        1   2.34s   76.22%  0.57%  0.392%  instance_sequProg_busyTime           FCU
4  scf        all 0.65s   21.17%  0.16%  0.109%  tot_asyncIO_busyTime                 -
5  scf        1   0.65s   21.17%  0.16%  0.109%  instance_asyncIO_busyTime            FCU
6  scf        all 2.99s   97.39%  0.73%  0.500%  tot_process_busyTime                 -
7  scf        1   2.99s   97.39%  0.73%  0.500%  instance_busyTime                    FCU
```

```
7a scf        1   3.07s   100.00% 0.75%  0.514%  tot_busyTime(corrected)              FCU
8  scf        all 3.07s   100.00% 0.75%  0.514%  tot_busyTime(system)                 -

1  scr        all 609.49s -       -      -       duration
2  scr        all 2.51s   73.82%  0.61%  0.420%  tot_sequProg_busyTime                -
3  scr        1   2.51s   73.82%  0.61%  0.420%  instance_sequProg_busyTime           FCU
4  scr        all 0.80s   23.53%  0.20%  0.134%  tot_asyncIO_busyTime                 -
5  scr        1   0.80s   23.53%  0.20%  0.134%  instance_asyncIO_busyTime            FCU
6  scr        all 3.31s   97.35%  0.81%  0.554%  tot_process_busyTime                 -
7  scr        1   3.31s   97.35%  0.81%  0.554%  instance_busyTime                    FCU
7a scr        1   3.40s   100.00% 0.83%  0.569%  tot_busyTime(corrected)              FCU
8  scr        all 3.40s   100.00% 0.83%  0.569%  tot_busyTime(system)                 -

1  sdmsp      all 615.98s -       -      -       duration
2  sdmsp      all 5.38s   66.01%  1.32%  0.900%  tot_sequProg_busyTime                -
3  sdmsp      1   5.38s   66.01%  1.32%  0.900%  instance_sequProg_busyTime           PSU
4  sdmsp      all 2.50s   30.67%  0.61%  0.418%  tot_asyncIO_busyTime                 -
5  sdmsp      1   2.50s   30.67%  0.61%  0.418%  instance_asyncIO_busyTime            PSU
6  sdmsp      all 7.88s   96.69%  1.93%  1.319%  tot_process_busyTime                 -
7  sdmsp      1   7.88s   96.69%  1.93%  1.319%  instance_busyTime                    PSU
7a sdmsp      1   8.15s   100.00% 1.99%  1.364%  tot_busyTime(corrected)              PSU
8  sdmsp      all 8.15s   100.00% 1.99%  1.364%  tot_busyTime(system)                 -

1  sdpyr      all 610.71s -       -      -       duration
2  sdpyr      all 3.71s   66.61%  0.91%  0.621%  tot_sequProg_busyTime                -
3  sdpyr      1   3.71s   66.61%  0.91%  0.621%  instance_sequProg_busyTime           FCU
4  sdpyr      all 1.73s   31.06%  0.42%  0.290%  tot_asyncIO_busyTime                 -
5  sdpyr      1   1.73s   31.06%  0.42%  0.290%  instance_asyncIO_busyTime            FCU
6  sdpyr      all 5.44s   97.67%  1.33%  0.910%  tot_process_busyTime                 -
7  sdpyr      1   5.44s   97.67%  1.33%  0.910%  instance_busyTime                    FCU
7a sdpyr      1   5.57s   100.00% 1.36%  0.932%  tot_busyTime(corrected)              FCU
8  sdpyr      all 5.57s   100.00% 1.36%  0.932%  tot_busyTime(system)                 -

1  sdt1       all 612.69s -       -      -       duration
2  sdt1       all 9.90s   73.01%  2.42%  1.657%  tot_sequProg_busyTime                -
3  sdt1       1   9.90s   73.01%  2.42%  1.657%  instance_sequProg_busyTime           FCU
3  sdt1       2   0.00s   0.00%   0.00%  0.000%  instance_sequProg_busyTime           FCU
3  sdt1       3   0.00s   0.00%   0.00%  0.000%  instance_sequProg_busyTime           FCU
```

```
3  sdt1       4   0.00s   0.00%   0.00%  0.000%  instance_sequProg_busyTime          PSU
4  sdt1       all 3.27s   24.12%  0.80%  0.547%  tot_asyncIO_busyTime                -
5  sdt1       1   1.44s   10.62%  0.35%  0.241%  instance_asyncIO_busyTime           FCU
5  sdt1       2   0.46s   3.39%   0.11%  0.077%  instance_asyncIO_busyTime           FCU
5  sdt1       3   0.46s   3.39%   0.11%  0.077%  instance_asyncIO_busyTime           FCU
5  sdt1       4   0.91s   6.71%   0.22%  0.152%  instance_asyncIO_busyTime           PSU
6  sdt1       all 13.17s  97.12%  3.22%  2.204%  tot_process_busyTime                -
7  sdt1       1   11.34s  83.63%  2.77%  1.898%  instance_busyTime                   FCU
7a sdt1       1   11.68s  86.10%  2.86%  1.954%  tot_busyTime(corrected)             FCU
7  sdt1       2   0.46s   3.39%   0.11%  0.077%  instance_busyTime                   FCU
7a sdt1       2   0.47s   3.49%   0.12%  0.079%  tot_busyTime(corrected)             FCU
7  sdt1       3   0.46s   3.39%   0.11%  0.077%  instance_busyTime                   FCU
7a sdt1       3   0.47s   3.49%   0.12%  0.079%  tot_busyTime(corrected)             FCU
7  sdt1       4   0.91s   6.71%   0.22%  0.152%  instance_busyTime                   PSU
7a sdt1       4   0.94s   6.91%   0.23%  0.157%  tot_busyTime(corrected)             PSU
8  sdt1       all 13.56s  100.00% 3.32%  2.270%  tot_busyTime(system)                -

1  sdtmp      all 611.29s -       -      -       duration
2  sdtmp      all 5.60s   70.44%  1.37%  0.937%  tot_sequProg_busyTime               -
3  sdtmp      1   5.60s   70.44%  1.37%  0.937%  instance_sequProg_busyTime          PSU
3  sdtmp      2   0.00s   0.00%   0.00%  0.000%  instance_sequProg_busyTime          PSU
4  sdtmp      all 2.06s   25.91%  0.50%  0.345%  tot_asyncIO_busyTime                -
5  sdtmp      1   1.20s   15.09%  0.29%  0.201%  instance_asyncIO_busyTime           PSU
5  sdtmp      2   0.86s   10.82%  0.21%  0.144%  instance_asyncIO_busyTime           PSU
6  sdtmp      all 7.66s   96.35%  1.87%  1.282%  tot_process_busyTime                -
7  sdtmp      1   6.80s   85.53%  1.66%  1.138%  instance_busyTime                   PSU
7a sdtmp      1   7.06s   88.77%  1.73%  1.181%  tot_busyTime(corrected)             PSU
7  sdtmp      2   0.86s   10.82%  0.21%  0.144%  instance_busyTime                   PSU
7a sdtmp      2   0.89s   11.23%  0.22%  0.149%  tot_busyTime(corrected)             PSU
8  sdtmp      all 7.95s   100.00% 1.94%  1.331%  tot_busyTime(system)                -

1  sysinit    all 616.59s -       -      -       duration
2  sysinit    all 1.40s   56.91%  0.34%  0.234%  tot_sequProg_busyTime               -
3  sysinit    1   1.40s   56.91%  0.34%  0.234%  instance_sequProg_busyTime          FCU
4  sysinit    all 0.57s   23.17%  0.14%  0.095%  tot_asyncIO_busyTime                -
5  sysinit    1   0.57s   23.17%  0.14%  0.095%  instance_asyncIO_busyTime           FCU
6  sysinit    all 1.97s   80.08%  0.48%  0.330%  tot_process_busyTime                -
7  sysinit    1   1.97s   80.08%  0.48%  0.330%  instance_busyTime                   FCU
```

```
7a sysinit    1   2.46s   100.00% 0.60%  0.412%  tot_busyTime(corrected)              FCU
8  sysinit    all 2.46s   100.00% 0.60%  0.412%  tot_busyTime(system)                 -

1  tmp        all 613.43s -       -      -       duration
2  tmp        all 17.44s  77.58%  4.27%  2.919%  tot_sequProg_busyTime                -
3  tmp        1   17.44s  77.58%  4.27%  2.919%  instance_sequProg_busyTime           PSU
3  tmp        2   0.00s   0.00%   0.00%  0.000%  instance_sequProg_busyTime           PSU
4  tmp        all 4.36s   19.40%  1.07%  0.730%  tot_asyncIO_busyTime                 -
5  tmp        1   2.39s   10.63%  0.58%  0.400%  instance_asyncIO_busyTime            PSU
5  tmp        2   1.97s   8.76%   0.48%  0.330%  instance_asyncIO_busyTime            PSU
6  tmp        all 21.80s  96.98%  5.33%  3.649%  tot_process_busyTime                 -
7  tmp        1   19.83s  88.21%  4.85%  3.319%  instance_busyTime                    PSU
7a tmp        1   20.45s  90.96%  5.00%  3.422%  tot_busyTime(corrected)              PSU
7  tmp        2   1.97s   8.76%   0.48%  0.330%  instance_busyTime                    PSU
7a tmp        2   2.03s   9.04%   0.50%  0.340%  tot_busyTime(corrected)              PSU
8  tmp        all 22.48s  100.00% 5.50%  3.762%  tot_busyTime(system)                 -

1  tmy_cychdl all 778.37s -       -      -       duration
2  tmy_cychdl all 14.77s  30.45%  3.61%  2.472%  tot_sequProg_busyTime                -
3  tmy_cychdl 1   14.77s  30.45%  3.61%  2.472%  instance_sequProg_busyTime           FCU
4  tmy_cychdl all 32.71s  67.44%  8.00%  5.475%  tot_asyncIO_busyTime                 -
5  tmy_cychdl 1   32.71s  67.44%  8.00%  5.475%  instance_asyncIO_busyTime            FCU
6  tmy_cychdl all 47.48s  97.90%  11.61% 7.947%  tot_process_busyTime                 -
7  tmy_cychdl 1   47.48s  97.90%  11.61% 7.947%  instance_busyTime                    FCU
7a tmy_cychdl 1   48.50s  100.00% 11.86% 8.117%  tot_busyTime(corrected)              FCU
8  tmy_cychdl all 48.50s  100.00% 11.86% 8.117%  tot_busyTime(system)                 -

1  tmy_reqhdl all 599.71s -       -      -       duration
2  tmy_reqhdl all 2.15s   70.49%  0.53%  0.360%  tot_sequProg_busyTime                -
3  tmy_reqhdl 1   2.15s   70.49%  0.53%  0.360%  instance_sequProg_busyTime           FCU
4  tmy_reqhdl all 0.81s   26.56%  0.20%  0.136%  tot_asyncIO_busyTime                 -
5  tmy_reqhdl 1   0.81s   26.56%  0.20%  0.136%  instance_asyncIO_busyTime            FCU
6  tmy_reqhdl all 2.96s   97.05%  0.72%  0.495%  tot_process_busyTime                 -
7  tmy_reqhdl 1   2.96s   97.05%  0.72%  0.495%  instance_busyTime                    FCU
7a tmy_reqhdl 1   3.05s   100.00% 0.75%  0.510%  tot_busyTime(corrected)              FCU
8  tmy_reqhdl all 3.05s   100.00% 0.75%  0.510%  tot_busyTime(system)                 -

1  usd        all 611.58s -       -      -       duration
```

```
2  usd        all 7.92s   75.50%  1.94%  1.326%  tot_sequProg_busyTime              -
3  usd        1   7.92s   75.50%  1.94%  1.326%  instance_sequProg_busyTime         FCU
4  usd        all 2.22s   21.16%  0.54%  0.372%  tot_asyncIO_busyTime               -
5  usd        1   2.22s   21.16%  0.54%  0.372%  instance_asyncIO_busyTime          FCU
6  usd        all 10.14s  96.66%  2.48%  1.697%  tot_process_busyTime               -
7  usd        1   10.14s  96.66%  2.48%  1.697%  instance_busyTime                  FCU
7a usd        1   10.49s  100.00% 2.57%  1.756%  tot_busyTime(corrected)            FCU
8  usd        all 10.49s  100.00% 2.57%  1.756%  tot_busyTime(system)               -

1  vgs        all 607.15s -       -      -       duration
2  vgs        all 2.60s   73.65%  0.64%  0.435%  tot_sequProg_busyTime              -
3  vgs        1   2.60s   73.65%  0.64%  0.435%  instance_sequProg_busyTime         FCU
4  vgs        all 0.87s   24.65%  0.21%  0.146%  tot_asyncIO_busyTime               -
5  vgs        1   0.87s   24.65%  0.21%  0.146%  instance_asyncIO_busyTime          FCU
6  vgs        all 3.47s   98.30%  0.85%  0.581%  tot_process_busyTime               -
7  vgs        1   3.47s   98.30%  0.85%  0.581%  instance_busyTime                  FCU
7a vgs        1   3.53s   100.00% 0.86%  0.591%  tot_busyTime(corrected)            FCU
8  vgs        all 3.53s   100.00% 0.86%  0.591%  tot_busyTime(system)               -

1  vgs_sv     all 601.47s -       -      -       duration
2  vgs_sv     all 2.63s   79.94%  0.64%  0.440%  tot_sequProg_busyTime              -
3  vgs_sv     1   2.63s   79.94%  0.64%  0.440%  instance_sequProg_busyTime         FCU
4  vgs_sv     all 0.62s   18.84%  0.15%  0.104%  tot_asyncIO_busyTime               -
5  vgs_sv     1   0.62s   18.84%  0.15%  0.104%  instance_asyncIO_busyTime          FCU
6  vgs_sv     all 3.25s   98.78%  0.79%  0.544%  tot_process_busyTime               -
7  vgs_sv     1   3.25s   98.78%  0.79%  0.544%  instance_busyTime                  FCU
7a vgs_sv     1   3.29s   100.00% 0.80%  0.551%  tot_busyTime(corrected)            FCU
8  vgs_sv     all 3.29s   100.00% 0.80%  0.551%  tot_busyTime(system)               -

1  wpa        all 609.36s -       -      -       duration
2  wpa        all 2.37s   74.76%  0.58%  0.397%  tot_sequProg_busyTime              -
3  wpa        1   2.37s   74.76%  0.58%  0.397%  instance_sequProg_busyTime         FCU
4  wpa        all 0.68s   21.45%  0.17%  0.114%  tot_asyncIO_busyTime               -
5  wpa        1   0.68s   21.45%  0.17%  0.114%  instance_asyncIO_busyTime          FCU
6  wpa        all 3.05s   96.21%  0.75%  0.510%  tot_process_busyTime               -
7  wpa        1   3.05s   96.21%  0.75%  0.510%  instance_busyTime                  FCU
7a wpa        1   3.17s   100.00% 0.78%  0.531%  tot_busyTime(corrected)            FCU
8  wpa        all 3.17s   100.00% 0.78%  0.531%  tot_busyTime(system)               -
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | wpp | all | 614.17s | - | - | - | duration | |
| 2 | wpp | all | 8.58s | 76.47% | 2.10% | 1.436% | tot_sequProg_busyTime | - |
| 3 | wpp | 1 | 8.58s | 76.47% | 2.10% | 1.436% | instance_sequProg_busyTime | FCU |
| 4 | wpp | all | 2.27s | 20.23% | 0.56% | 0.380% | tot_asyncIO_busyTime | - |
| 5 | wpp | 1 | 2.27s | 20.23% | 0.56% | 0.380% | instance_asyncIO_busyTime | FCU |
| 6 | wpp | all | 10.85s | 96.70% | 2.65% | 1.816% | tot_process_busyTime | - |
| 7 | wpp | 1 | 10.85s | 96.70% | 2.65% | 1.816% | instance_busyTime | FCU |
| 7a | wpp | 1 | 11.22s | 100.00% | 2.74% | 1.878% | tot_busyTime(corrected) | FCU |
| 8 | wpp | all | 11.22s | 100.00% | 2.74% | 1.878% | tot_busyTime(system) | - |
| - | - | - | - | - | - | 68.427% | totalSystemUtilisation_mode= | ONETARGET |
| - | - | - | - | - | - | 41.991% | systemUtilisation_(estimated)_for_node | FCU |
| - | - | - | - | - | - | 26.442% | systemUtilisation_(estimated)_for_node | PSU |

### 7.8.4 The Timer Report

The timer report also reflects the increased load and the higher number of periodic events. Significantly, the maen queue length is increased which implies an increased effort for manangement of the queue.

```
+-----------------------------+
+           Timer Report      |
+-----------------------------+


#TimerResponse:          13969
#TimerAutoRepetitions:   13248
#TimerRequest:           17068
#TimerDelete:            1790
#TimerInsert:            15277
MaximumQueueLength:      161
#QueueSamples:           31037
MeanValueOfQueueLength: 84.01
```

## 7.9 The "Error Injection" Case

For error injection the system processes were exposed to automated injection of errors except for sysinit, cmdhandler, ethdae and mildae. "sysinit" is only involved in system initialisation, cmdhandler, ethdae and mildae are involved in periodic commanding and therefore shall not be subject of error injection Therefore the remaining processes only shall be subject of error injection. For the selected mode this means that no output command is issued.

This will cause missing responses to which especially the initialsation procedure is extremely sensitive. This is well known by the project and will probably not be changed.

Nevertheless, a number of runs have been performed with an error injection probability increasing continuously from 0 to 1.

There is an interesting observation: the way the coverage of command lines and states changes w.r.t to the error injection probability: In the MSL case and due to the high sensitivity of the initialisation procedure against missing responses there is a sudden and significant decrease of the coverage figures.

Consequently, the change of the coverage vs. the change of the error injection probability is a measure for the robustness of the system against a certain type of errors which are injected: the smoother the change of the coverage is and the higher it is for higher error injection probabilities the more robust is the system against errors of the injected type.

Vice versa, the sooner the coverage decreases the less robust is a system.

The table on the next page gives the figures in case of MSL.

This table also includes the number of injected errors and the number of exceptions which occured in order to demonstrate that these figures are not an indicator for a system's robustness.

Clearly, the less command lines are executed (covered) the less is the number of exceptions and injected errors. So the number of exceptions and injected errors varies randomly more or less.

The figures for the coverage for the operational case were compared with the figures in case of activated error injection, but error injection probability 0. Both figures are equivalent.

| Error Injection Probability | Coverage of Command Lines %s | Coverage of States % | # Inejcted External Commands | # Injected Errors | # Exceptions |
|---|---|---|---|---|---|
| 0 | 95.94 | 100 | 1380 | 0 | 200 |
| 0.01 | 94.46 | 100 | 900 | 187 | 129 |
| ß.02 | 78.37 | 78.38 | 1340 | 230 | 212 |
| 0.03 | 78.64 | 78.38 | 1140 | 284 | 186 |
| 0.04 | 44.04 | 37.94 | 1120 | 148 | 93 |
| 0.05 | 44.04 | 37.84 | 1120 | 176 | 120 |
| 0.10 | 44.04 | 37.84 | 1140 | 270 | 270 |
| 0.50 | 41.81 | 36.04 | 1140 | 825 | 355 |
| 0.70 | 41.55 | 36.04 | 1120 | 1094 | 557 |
| 1.00 | 41.28 | 36.04 | 1120 | 1316 | 840 |

Table 7-1: Coverage vs. Error Injection Probabiity

# 8. Conclusions

## 8.1 General Conclusions

A number of verification and validation activities have been performed for the MSL software system. The approach was based on a formal specification of behaviour by means of Finite State Machines and extensions which increased the degree of formality to specify a system's behaviour. Also, the specification of consumed system resources and distribution was formalised.

Due to this formalisation a number of checks could be performed at pre-run-time which identified already a number of errors like unreachable or missing code or conflicting allocation of CPU's.

Then at run-time more errors could be detected by built-in error checks which identified e.g. wrong inputs (commands not expected in the actual state or commands sent to the wrong instance of a process).

Finally, at post-run-time more errors could be identified by the generated MSC's, timing diagrams and reports on coverage of command lines and states.

In principle, at each of the three phases a certain type of errors can be detected by the means available for this phase.

Also, it was recognised that some extensions made for operational reasons or for higher user comfort decreased the degree of formality so that less checks could be made and errors could be detected at pre-run-time. Such errors were detected at run-time or post-run-time.

Some of the checks may be made available again at pre-run-time by increased analysis effort.

The same is true for test automation and automated stimulation of processes with commands.

The generated performance figures gave a good insight view on the system performance. Except for CPU utilisation - which is still an open issue which will be covered when the application has been ported to the target - the provided performance figures indicate that the system is not critical in such areas like network utilisation and command buffer utilisation.

The available implementation and infrastructure for analysis of system properties may be used for the following phases of the project's lifecycle either on the host or on the target system.

The execution of the "operational case" and the analysis of the results demonstrated that the current system definition is consistent, correct and complete. However, due to missing functionality some properties could not be tested because they were not made visible for the time being: the explicit expression of system impacts on command line level were considered as too complex.

E.g. the dependency of system behaviour on housekeeping data may be very complex. Therefore it was suggested to think about how the complex nature of such decisions could be mapped on a number of simple decisions which are appropriate for FSM and command line level. This would allow to verifiy and validate the system to a much higher degree.

Due to the formal specification by FSM's it was possible to automate the generation of the command dispatcher completely. This possibility was identified by the MSL project and taken into account for its sofware design. Only by adding little information to a command line - the mapping of O/G commands onto O/B commands, the timeout value (if any) and the sequence number - the command dispatcher software can be constructed automatically. Of course, this automation implies cost saving for future projects.

The construction of the command dispatcher may be compared with processing of a matrix. If no information about a potential symmetry is available all elements have to be processed. However, if by

some formal rules it can be proven that the matrix is symmetric, about half of the effort for processing can be saved.

Similarly, by introduction of formality the correlation between the behaviour of the processes which receive a (number of) ground commands and the dispatcher becomes visible. This allows to apply some construction rules for the command dispatcher, to make it generic and to automate the generation of the specific dispatching software.

## 8.2 Specific Conclusions

The MSL software system turned out to be rather complex system and it was subject of different V&V activities:

1. static checking
2. checking of the operational case from system initialisation to normal operations
3. checking under stress conditions by automated stimulation with external (ground) and internal (on-board) commands
4. checking under error injection conditions.

By (1) and (2) it could be proven that the system will behave correctly for the current specification and implementation and that there are sufficient resources (with the open issue on CPU utilisation, see above).

By (3) it was shown that the system does not crash in case of an overload situation. It is up to the project to decide if more self-protection mechanisms are needed or not, and to prevent exhausting of command buffers.

By (4) it was shown how the system behaves in case issueing of commands is partially suppressed. In general, it can be concluded that the system is sensitive to loss of command messages. This knowledge is used by the project to initiate recovery mechanisms on application software level.

From an operational point of view it was identified that external commanding is possible when mildae and ethdae have been initialised which happens at a very early stage, but it is not desired because the other processes may not be initialised or not ready to accept external commands. It is also up to the project to decide if something has to be changed regarding such observations.

## 8.3 Assessment on the V&V Environment

The applied V&V environment is strongly related to the ISG approach, the provided infrastructure and the command procedure table which forms the base for generation of the software, the reporting and automated generation of the graphical diagrams.

From a user's point of view a learning phase was needed at the beginning to understand the approach, and to learn how the elements for system description can and need to be applied. Similarly, a user needs to learn how he can take benefit from the provided means like error messages, reports and graphical support tools.

If he is familiar with such supporting facilities it is rather easy to perform verification and valdiation in this environment and to update the system definition when an error is identified.

Although a number of structural changes were needed this did not cause a lot of effort. Most of the changes could be done within minutes, 15 minutes are already very high, and 30 minutes is very unlikely and was only needed when a number of similar constructs needed to be updated. After the update it took about one hour (to wait) to get new results.

Also, the type and number of errors found at pre-run-time, run-time and post-run-time are an indication of how well the set up of the system definition can really be done: no very serious errors were detected. About 50% of the errors were related to copy-paste of command lines and forgotten changes. A minor part was related to the complexity of the required behaviour, e.g. when more than one process level was involved to provide a feedback, i.e. a process receiving a command has to involve another process before it can respond. Another erorr type was that it just was forgotten to respond or the wrong destination was given.

Moreover, it was very easy to establish a sequence of states to lock against unforeseen events and to initiate error handling. E.g. for the initialisation procedure 35 states were introduced and the sequence was changed several times.

Therefore it may be concluded that the use of a formal method, introduced by the structure of the command procedure table which is based on FSM's, prevented already serious bugs.

The current system definition is based on a textual notation which may be dissatisfying for some users. In a future version it may be based on graphical notations. However, it shall also be mentionned that the textual notation - as it is true for all textual notations - implies a high density of included information compared with graphical notations.

For another project a user established 17 lines to express the behaviour of a system component including error injection and timeout condtions. He needed one page and seven state diagrams to roughly describe what is included in the command procedure lines, not covering matters of error injection, timeout conditions, distribution and performance.

On the other side, such graphical state diagrams may be helpful to get an idea on a system's behaviour prior to generation of the command lines.

Also, it has been recognised that much more time is needed (about 10 to 20 times) to identify bugs in a conventional C environment.

Another aspect is the identification of the robustness of a system against injected errors. The higher is the percentage of covered command lines and covered states at a certain error injection probability, or vice versa the higher is the error injection probability for a given coverage, the more robust is a system regarding occurence of errors. So by a number ot test runs with an increasing probability of error injection and monitoring of the coverage figures the robustness of a system can be identified.

If the coverage decreases suddenly at a low error injection probability there is a high sensitivity for faults.

In case of MSL this is true for the initialisation procedure, but it is also true that the MSL system shall be aborted in case an error occurs during this start up sequence.

Finally, an essential point of the V&V approach is that it was possible to identify errors within the rather complex specification (37 processes, abouit 230 commands, 111 states inside the processes in total) WITHOUT having any specific knowledge on how the system shall work. Just by applying a formal notation and formal checks on it (including analysis of coverage) it was possible that the V&V environment flagged an error.

This "automated" response by the tool is important for two reasons:

- firstly, an engineer who does apply the V&V approach does not need to be familiar with system details,
- secondly, the engineer does not need to ask the V&V environment whether a certain property is available or not, the V&V environment automatically informs about a weakness or an error.

Hence, it is not possible to forget to ask and consequently not to be informed about an error.

Last but not least, it is considered as an advantage that the system and the V&V environment is completely portable which allowed to perform the V&V activities within the more powerful UNIX platform.

## 8.4 Final Remarks on Target Platform Activities

When the draft final report was written it was expected to port the existing software from the Sparc/Unix to the SPLC/VxWorks platform within about two weeks. However, about two months were needed for the following resasons:

- A number of functions needed to be added to get the support which is inherently provided by a Unix platform.

  In part, such functions were needed for provision of a complete sete of reports as it was done on the Unix platform.

- A number of functions needed to be added to get better visibility on what the system does or (more important) what the system did before it crashed.

  Due to the global address space a VxWorks system is very sensitive against addressing faults. Therefore in most cases a post-mortem analysis was not possible and a facility needed to be added which provided a trace up to the last executed statement before the crash occured.

  Due to the complexity of the system (about 40 processes) and their interaction more complex sequences required better tracing capabilities than a debugger provides.

  A significant disadvantage is that in some cases no tasking environment is available and only a very small subset of OS functions can be used. This limits the visibility and makes it difficult to identify a problem (not necessarily) a bug in such parts

- Such parts of the software which are only related to the VxWorks platform needed to be tested and integrated under the conditions described above. This required significantly higher effort compared with the Unix platform.

  Special care needed to be applied for memory allocation (only once per system and not once per process) and for racing conditions which are characteriostic for a RT OS.

- The time was not provided at a resolution sufficient for performance measurements.

  This required implementation of an own high-resolution timer.

- Two bugs were found for the co-processor of the SPLC which were difficult to identify.

  A crash was observed when the memShow function was called from within a user C function. This was also difficult to identify.

- A first iteration was performed to reduce the required amount of memory. This addressed the increase of a higher degree of generic functions and introduction of a strategy for stack minimisation and minimisation of the memory allocated by malloc. This saved about 1 MB for the whole application.

Nevertheless, above problems have been solved and they do not occur again in future.

Concerning performance the following conclusions can be drawn:

- The timing and sizing budgets - as measured currently - indicate that the full memory and the CPU power will probably be needed. It may even happen that a CPU overload occurs. In this case the project will extend the current timing constraints. The probability that the current resource limits will be exceeded is not high, but has to be taken into account. However, it seems that the overall performance can still be met even for such a worst case.

- For a higher number of processes the memory consumption by code and static data becomes less important - especially when it has already been minimised - while consumers like stack and communication buffers become more important.

# 9. Recommendations and Guidelines

According to the current experience made by the MSL project and two more (smaller) applications the following recommendations and guidelines are given:

**Start-Up and General Rules**

- A user needs to become familiar with formal specification, with the idea of ISG and system definition by the command procedure table.

  As for any other language a certain learning curve occurs.

- A user shall start incrementally with system definition to get an early and immediate feedback on what he has specified

- A user should already have an (rough) idea on system communication, the distribution and the network and the sytem components.

  Of course, this input may be changed due to the feedback provided by the V&V environment, but at least a rough idea is needed to write the command lines and to express behaviour and communication.

- A user should apply as far as possible states and state transitions to protect against inadvertent multiple inputs of the same command

  If e.g. it is forbidden that a command is received twice during a cycle a state transition after the reception of the first input will lock further inputs and open the possibility for exception handling.

- A user needs to be carefully when he introduces certain definitions and properties which cannot formally be checked because they cannot be limited by genral rules.

  This is especially true for:

  -- copy-paste operations for command lines

     a user shall not forget to change all relevant fields

  -- definition of the destination

     If the destination process is given explicitly, the instance number must be given correctly.

     Think about if a fixed number needs to be given or "0" to take the actual instance number of the sending process.

  -- for a group of command lines a state transition can only be initiated by the last command line

     (an earlier state transition would lead to non-reachable code and is therefore forbidden)

  -- define an event / condition to start periodic activities

  -- do not forget to specify an exception handler for a timeout condition

     (otherwise the checktbl-tool will remind you) and to activitate automated reset of timeout-monitoring if you do not ask for automated generation of such lines.

  -- define the network topology and its performance in file easysystem.def

**Rules for the V&V Activities**

- apply the following steps of verification and validation

  -- apply the "checktbl" tool for static analysis of the command procedure contents

     read carefully the error reports and remove the errors

structural errors (may) lead to generation of erroneous code which is rejected by the compiler later on or the system may crash at run-time. For these reasons the implementation script "createapplfiles" aborts a run in case of major errors.

-- then start the implementation script "createapplfiles" after having also provided and updated the other two files (beside the command procedure table cmdproc.in): easysystem.def and easyconf.h

-- look carefully for warning or error messages issued by createapplfiles or scripts and systems services called by it. Especially, this is needed for the first run of a command procedure table or a major change of it.

-- for the first steps take higher values for timeout conditions in order not to disturb the operations by undesired timouts. If the system is working properly set the values back to the realistic values.

-- in order to avoid undesired high system load due to errors e.g. in the communication part take at the beginning higher values for the cycle periods. Later on the real values can be inserted.

-- execute the generated system and wait until the evaluation report is available and the texteditor window opens for isgeval.log

look on the coverage figures and which command lines and states are not covered

look for error messages in error.log or grep for "*** ERROR" in file alog or alogr*

identify the MSC-file by "ls *.msc", open the textfile and search for "ERROR"

look into the expanded command procedure table included in file "cmdprocin.in" to understand deviations from expected behaviour

use "starttime" as timestamp to search for correlated events, especially for a search in the msc/mscp-files

make use of the extended MSC tracing capabilities which generate information about initial and final states, correlation with command lines and display the contents of commands including the attached data with the mscope-tool for display of timing diagrams

in case more support is needed use the graphical tools for MSC's (mscviewer) and timing diagrams (mscope) to understand the control flow

look on the utilisation of the command buffer, a high mean queue length may indicate a problem or an overload

look on CPU and network utilisation in order to detect early a problem with system resources

possibly you need to change the distribution to balance the load between the CPU's (however this may increase network utilisation)

look on response times, if they are in the range you are expecting, if not if this can be understood (e.g. by load imposed by instrumentation for reporting).

Look on timer utilisation, if it is high, can it be reduced, did you place timeouts into activities which run at high frequencies, if this can be avoided without compromising the system's robustness, do it

introduce deadline monitoring if needed (remember that monitoring of overruns is inherently done by the system)

after each identification of a bug update the table and perform the next run until the desired behaviour, coverage and performance is achieved

**Rules for Stress Testing and Error Injection**

-- when the system shows the desired properties in the operational case, apply stress testing

define which processes shall not be subject of automated stimulation in file "easysystem.def"

start "createapplfiles" and build a new system which is exposed to stress testing

identify errors which occur and decide if they are relevant or not

monitor the command buffers and look if and why they are exhausted, decide if you need to protect against overloads or not

-- apply error injection

define a probability for error injection in file "easysystem.def"

define which processes shall not be exposed to error injection in file "easysystem.def"

look on the achieved coverage of command lines and states

vary the probability figure from close to 0 up to 1 and analyse the coverage figures

a sudden decrease at low probability indicates high sensitivity against errors

if this occurs, decide if you need to improve your system