VTT TIEDOTTEITA – MEDDELANDEN – RESEARCH NOTES 1933

Developing application frameworks for mission-critical software

Using space applications as an example

Tuomas Ihme

VTT Electronics

Pekka Kumara & Keijo Suihkonen

Patria Finavitec Oy Systems

Niklas Holsti & Matti Paakko

Space Systems Finland Ltd.

TECHNICAL RESEARCH CENTRE OF FINLAND

ESPOO 1998

ISBN 951–38–5401–9 (soft back ed.)
ISSN 1235–0605 (soft back ed.)

ISBN 951–38–5402–7 (URL: http://www.inf.vtt.fi/pdf/)
ISSN 1455–0865 (URL: http://www.inf.vtt.fi/pdf/)

JULKAISIJA – UTGIVARE – PUBLISHER

Valtion teknillinen tutkimuskeskus (VTT), Vuorimiehentie 5, PL 2000, 02044 VTT
puh. vaihde (09) 4561, faksi (09) 456 4374

Statens tekniska forskningscentral (VTT), Bergsmansvägen 5, PB 2000, 02044 VTT
tel. växel (09) 4561, fax (09) 456 4374

Technical Research Centre of Finland (VTT), Vuorimiehentie 5, P.O.Box 2000, FIN–02044 VTT, Finland
phone internat. + 358 9 4561, fax + 358 9 456 4374

VTT Elektroniikka, Sulautetut ohjelmistot, Kaitoväylä 1, PL 1100, 90571 OULU
puh. vaihde (08) 551 2111, faksi (08) 551 2320

VTT Elektronik, Inbyggd programvara, Kaitoväylä 1, PB 1100, 90571 ULEÅBORG
tel. växel (08) 551 2111, fax (08) 551 2320

VTT Electronics, Embedded Software, Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland
phone internat. + 358 8 551 2111, fax + 358 8 551 2320

Technical editing Maini Manninen

LIBELLA PAINOPALVELU OY, ESPOO 1998

Ihme, Tuomas, Kumara, Pekka, Suihkonen, Keijo, Holsti, Niklas & Paakko, Matti. Developing application frameworks for mission-critical software. Using space applications as an example. Espoo 1998, Technical Research Centre of Finland, VTT Tiedotteita– Meddelanden – Research Notes 1933. 92 p. + app. 20 p.

Keywords computer systems programs, software, space software, object-oriented analysis, object-oriented design

Abstract

In this research note the key results from the Space2000/sw project are presented. The main goal of the project was to evaluate and adapt modern software methodologies for mission-critical applications using space software as an example.

This report introduces an outline for a set of object- and component-oriented software development activities for mission-critical software using the on-board control software of X-ray spectrometers as an example. The approach emphasises the use of software architectures, design patterns and application frameworks.

Modern real-time software methodologies consist of an integrated set of methods, CASE tools and informal, semi-formal and formal description techniques. The CASE tools support the simulation and validation of formal design models and components as well as automatic generation of target code from the design models. This report introduces an evaluation framework for such methodologies. The framework includes a hierarchical set of criteria that emphasises the development needs of mission-critical space software. The evaluation framework was used in the evaluations of real-time software methodologies. The evaluations are described in this report. The evaluated methodologies are based on OMT (Object Modelling Technique), MSC (Message Sequence Charts), statecharts and SDL (Specification and Description Language) notations.

This report also includes a study of product data management (PDM) in the context of mission-critical applications using space applications as an example.

Preface

This report presents the key results of the work that has been done in the Space2000/sw project from November 1996 to September 1998. The project was carried out as a part of the Space 2000 technology programme (Appendix C), funded by the Technology Development Centre of Finland (TEKES). CCC Systems Oy, Patria Finavitec Oy Systems, Space Systems Finland Ltd., and VTT Electronics participated in the project.

The industry operating in mission-critical software business is presently showing great interest in formal and behavioural description techniques to be used for consistency checking and validating specifications and designs, as well as for automatic generation of code from design models. Several commercial software methodologies involve description techniques like these. Software development technologies usually have to be tailored to the specific characteristics and needs of individual mission-critical application domains. The software methodologies were evaluated and adapted in the Space2000/sw project for various mission-critical applications using space applications as an example.

I wish to thank the following steering group members of the Space2000/sw project for controlling and supervising the project, and giving invaluable comments and advice on topics related to this report: Kimmo Ahola (TEKES), Niklas Holsti (Space Systems Finland Ltd.), Pekka Kumara (Patria Finavitec Oy Systems), Olli Lehtoranta (CCC Systems Oy), Matti Paakko (Space Systems Finland Ltd.), Timo Pastila (Space Systems Finland Ltd.), Jyri Pyrrö (CCC Systems Oy), and Jorma Taramaa (VTT Electronics).

I am grateful to the collaboration team in the project for evaluating modern software methodologies from the point of view of various mission-critical space applications; the team members were Niklas Holsti, Pekka Kumara, Matti Paakko, and Keijo Suihkonen (Patria Finavitec Oy Systems). Without the collaboration of these persons this work would not have been possible. The team authored several technical reports and conference papers related to this report. I also wish to thank the project team members Jorma Taramaa and Jukka Toivanen at VTT Electronics for their excellent work on modelling experimentation and product data management during the project.

The financial support provided by TEKES and VTT is gratefully acknowledged.

Oulu, September 1998

Tuomas Ihme

Project manager

Contents

Abstract 3

Preface 4

1. Introduction 8

2. Object-oriented development of mission-critical software 10
   - 2.1 System Requirements Specification 10
   - 2.2 System Architecture Definition 14
   - 2.3 Subsystem analysis 15
   - 2.4 Subsystem design 17
     - 2.4.1 Architectural and behavioural design 17
     - 2.4.2 Data and algorithm modelling 19
     - 2.4.3 Concurrency of processes 19
     - 2.4.4 Hardware interface 20
     - 2.4.5 Testing 20
   - 2.5 Discussion 21
   - 2.6 Summary 22

3. Component-oriented development 23
   - 3.1 Introduction to domain analysis 23
   - 3.2 Introduction to SDL patterns and frameworks 24
   - 3.3 The spectrometer domain example 25
   - 3.4 The Measurement Control architectural pattern 31
     - 3.4.1 Intent 31
     - 3.4.2 Motivation 32
     - 3.4.3 Participants in the pattern 33
     - 3.4.4 Data Acquisition Control 33
     - 3.4.5 Data Management 34
   - 3.5 SDL and ROOM frameworks for the spectrometer controller family 36
     - 3.5.1 Interfaces for spectrometer controllers 36
     - 3.5.2 A ROOM framework for the spectrometer controller family 39
     - 3.5.3 Reusability of the SDL framework 40
     - 3.5.4 Documentation of the framework variability 43
     - 3.5.5 Dependence between component systems in the frameworks 43
   - 3.6 Discussion 45
   - 3.7 Summary 47

4. Evaluation of functional aspects of real-time methodologies 49
4.1 Introduction to the evaluation of functional aspects 49
4.2 The evaluation framework 50
4.2.1 The structure of the evaluation criteria 50
4.2.2 Capability profiles of the methodology 51
4.3 Support for the OORT method 52
4.3.1 Requirement Analysis 53
4.3.2 Architectural Design 54
4.3.3 Detailed Design 54
4.3.4 Test Design 54
4.3.5 Targeting 54
4.3.6 Testing 55
4.3.7 The OORT iterative process 55
4.3.8 Methodology evolution 55
4.4 ESA standards 55
4.4.1 Product standards 56
4.4.2 Process standards 56
4.5 Application requirements 56
4.6 Summary 57

5. CASE Tool Evaluation 58
5.1 Introduction 58
5.2 Capability profiles of the tools 59
5.3 Ease of use 61
5.3.1 Tailoring 62
5.3.2 Intelligence and Helpfulness 62
5.3.3 Predictability 62
5.3.4 Error Handling 62
5.3.5 Multiple Users 62
5.3.6 System Interface 63
5.4 Power 63
5.4.1 Tool Understanding 63
5.4.2 Tool Leverage 63
5.4.3 Tool State 63
5.4.4 Performance 64
5.5 Robustness 64
5.5.1 Consistency of the operation of the tool 64
5.5.2 Evolution 64
5.5.3 Fault Tolerance 64
5.5.4 Self Instrumentedness 64
5.6 Functionality 65
5.6.1 Methodological Support 65

5.6.2 Correctness 67
5.7 Ease of Insertion 67
5.8 Quality of Support 67
5.9 Discussion 67
5.10 Summary 68

6. Product data management 70
6.1 Introduction to product data management 70
6.1.1 PDM user functions 70
6.1.2 PDM utility functions 72
6.1.3 Benefits of PDM 72
6.2 PDM and mission-critical applications 73
6.2.1 Characteristics of the product 73
6.2.2 Characteristics of the production process 77
6.2.3 Product, process and PDM components 78
6.2.4 Typical lifespan of a mission-critical application development project 80
6.2.5 Focus areas of PDM in mission-critical application development 82
6.3 PDM tools for mission-critical applications 83
6.3.1 Product data management and product data models 83
6.3.2 PDM tools 85

7. Conclusions 87

References 89

APPENDICES
Appendix A: A Framework for the Evaluation of CASE Tools
Appendix B: Evaluation Questions for CASE Tools
Appendix C: Space 2000 - Space Equipment Technology 1996-2000

1. Introduction

Embedded computer systems, incorporated in various types of products and systems, are common in a wide range of everyday commodities as well as industrial and scientific equipment. Those systems are often mission-critical applications.

Most of the industrially used real-time system methods address only forward engineering needs, by the identification and refinement of data processing functions needed for responding to the real-time stimuli of the system environment. The functions are usually described using structured models based on data flow and state transition notations. The so-called structured real-time methods have been quite widely used since the mid-eighties.

Object-oriented techniques, frameworks, components, architectures and design patterns are better means for reusable mission-critical software assets. Application frameworks [Fayad & Schmidt 1997] are reusable semi-complete applications for the development of a family or families of similar applications in a specific application domain. The designer builds a new application or a part of it by specialising a framework. Yet, organised reuse of mission-critical software assets gained from the development of certain system families is not a widespread phenomenon.

Although there are several commercial general-purpose object-oriented methods and tools available, these seem inadequate to manage the complexity of mission-critical software. Although there are some commercial object-oriented methods for real-time systems available, these have only been used to a limited extent for mission-critical software. This is due to the following reasons, for example:

- The tool support for the method is unreliable due to a small number of users, e.g. the Hood method [Robinson 1992],

- No commercial tool support is available, e.g. the Octopus [Awad et al. 1996], Gomaa’s [Gomaa 1993], and Ellison’s [Ellison 1994] methods,

- Some methods are very new, e.g. Real-Time UML [Douglass 1998],

- CASE tools are often very expensive, and

- The selection of a CASE tool that best match the needs of an individual organisation is often a rather complicated and confusing process.

Industry is presently showing great interest in formal and behavioural description techniques such as SDL (Specification and Description Language) and ROOM (Real-

Time Object-Oriented Modeling) [Selic et al. 1994], to be used for consistency checking, validating specifications and designs and for automatic generation of code from design models.

Object-oriented techniques and notations such as OMT (Object Modelling Technique) and MSC (Message Sequence Charts) provide means for reusable embedded software. Many object-oriented features are already incorporated in the ROOM language and in the new SDL standards. The ROOM method includes the MSC notation. The ObjecTime tool by ObjecTime Limited supports the ROOM method. Some commercial methodologies, such as ObjectGEODE by Verilog SA and SDT by Telelogic AB, integrate OMT, MSC and SDL techniques.

The following commercial methodologies were evaluated and used during the Space2000/sw project: ObjectGEODE by Verilog SA, SDT by Telelogic AB and ObjecTime by ObjecTime Limited. The ObjectGEODE methodology consists of the OORT method [OORT 1996] and the ObjectGEODE Version 1.1 tool. The SDT methodology includes the SOMT method [SOMT 1996] and the SDT Version 3.11 tool. The ObjecTime methodology consists of the ROOM method [Selic et al. 1994] and the ObjecTime Version 4.4 tool.

Chapter 2 of this report outlines the object-oriented development of mission-critical software. Chapter 3 is concerned with the component-oriented development, including domain analysis and the development of SDL patterns as well as SDL and ROOM frameworks. The design models presented in Chapter 2 are used in Chapter 3. Chapter 4 describes the evaluation of the functional aspects of real-time methodologies, using the ObjectGEODE methodology as an example. Chapter 5 describes the evaluation of the CASE tools that support the methods discussed in the previous chapters. Chapter 6 takes a look at product data management in context of mission-critical space applications.

Finally, Chapter 7 concludes what has been achieved and outlines the future development and research problems in the field. The appendices include the evaluation framework and evaluation questions for CASE Tools as well as an abstract of the Space 2000 technology programme.

The SIXA on-board software [Toivanen 1995, Leppälä et al. 1996] is used as example of mission-critical software particularly in Chapters 2 and 3.

2. Object-oriented development of mission-critical software

Commercial object-oriented technology usually has to be tailored to the specific characteristics and needs of individual embedded computer system domains. This chapter is concerned with the object-oriented development of mission-critical software, using scientific on-board X-ray spectrometer control software as an example. An outline of a set of object-oriented software development activities for X-ray spectrometers will be given.

This chapter will focus on the applicability, and tailoring needs, of the OORT, SOMT and ROOM methods to the object-oriented modelling of the SIXA on-board software [Toivanen 1995]. The experiences gained in the evaluation of the methods will be discussed. Chapter 5 will describe the evaluation of the CASE tools that support the methods.

The development and documentation needs of the existing SIXA software are used as the basic evaluation criterion of the methods. The SIXA instrument is a multielement X-ray photon counting spectrometer. A summary of the object-oriented development of X-ray spectrometer software has been published in [Ihme 1998a].

Examples of the use of architectural patterns [Buschman et al. 1996, Douglass 1998] will be shown. The presented examples are derived from the documentation of SIXA software. In the construction of the example models, some of the SIXA operations were simplified in order to keep the size of the model small. The most important simplification is that the examples include measurement functions of the SIXA instrument only, thus excluding housekeeping functions, for example. The system is called SIXA Measurement Controller hereafter.

2.1 System Requirements Specification

The product requirements specification of embedded systems often ignores software. Software development is only one of the subprojects of the master project. The system requirements specification recaptures the relevant product requirements for the software in a way that supports software development. The following models may be needed for the system requirements specification document of the SIXA Measurement Controller software:

- Textual Requirements,
- Hardware Architecture,

- Concept Dictionary,
- Requirements Use Case Model and
- System Context Diagram

Hardware architecture views show the hardware components and interfaces with which the software relates. The SOMT method provides a good example of the concept dictionary (data dictionary).

The requirements use case model may include a list of actors and use cases and a number of MSCs [SOMT 1996]. Use cases are first recorded using natural text structured into a number of text fields. MSC scenarios are then created from these textual use cases. Two MSC scenarios for the SIXA Measurement Controller are shown in Figures 1 and 2, depicting typical observation sequences of the controller:

- Receive measurement parameters
- Repeat for each target:
    - Switch on analog electronics
    - Receive target coordinates
    - Start observation time
    - Finish observation time
- Begin ground contact
- Transmit science data

The operating modes of the controller are Standby, Measurement, Data Retention and Dumping.

The interaction relationships between the SIXA Measurement Controller and external actors from the point of view of the software to be developed are shown in Figure 3. The SIXA Measurement Controller is installed in a satellite and controlled by a ground station via the satellite. The controller is responsible for three energy-spectrum observing modes and three single event characterisation observing modes. An array detector is used for detecting X-ray photons.

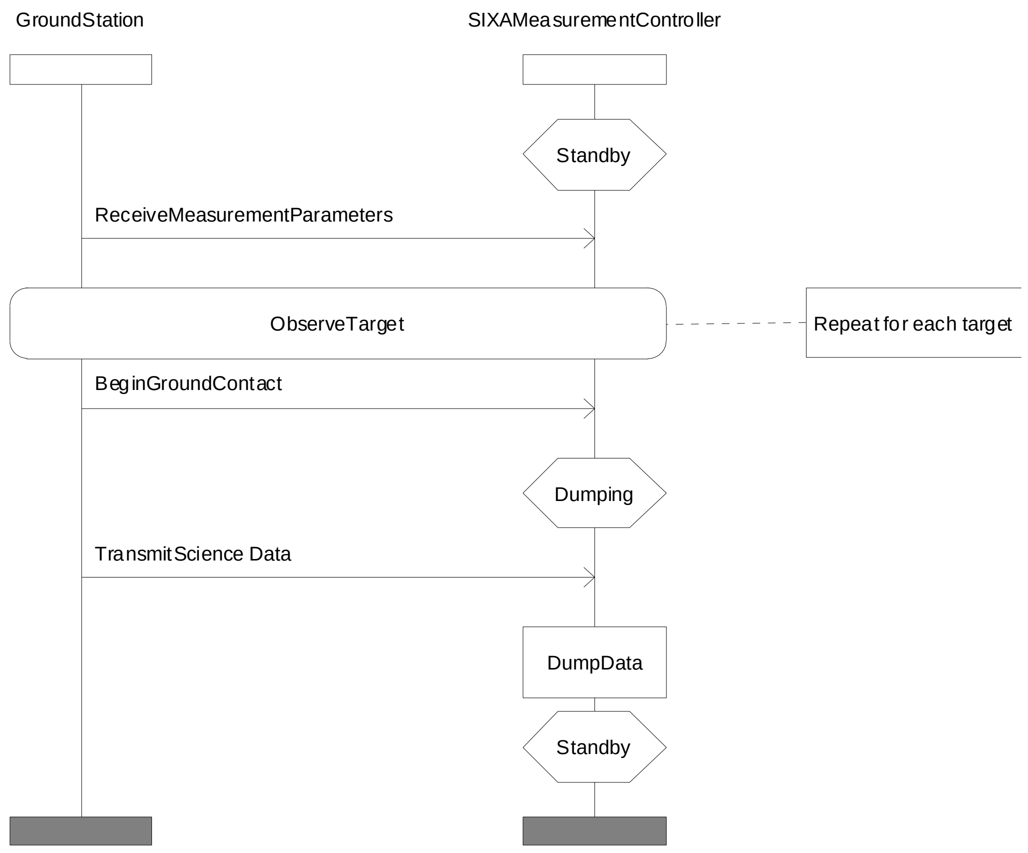


Figure 1. The Observation scenario of the SIXA Measurement Controller.

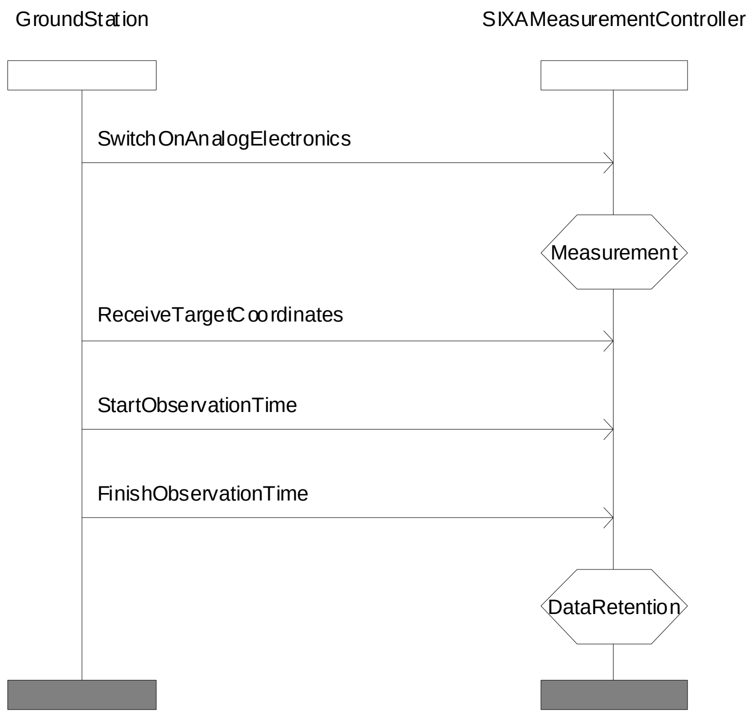


Figure 2. The Observe Target scenario.

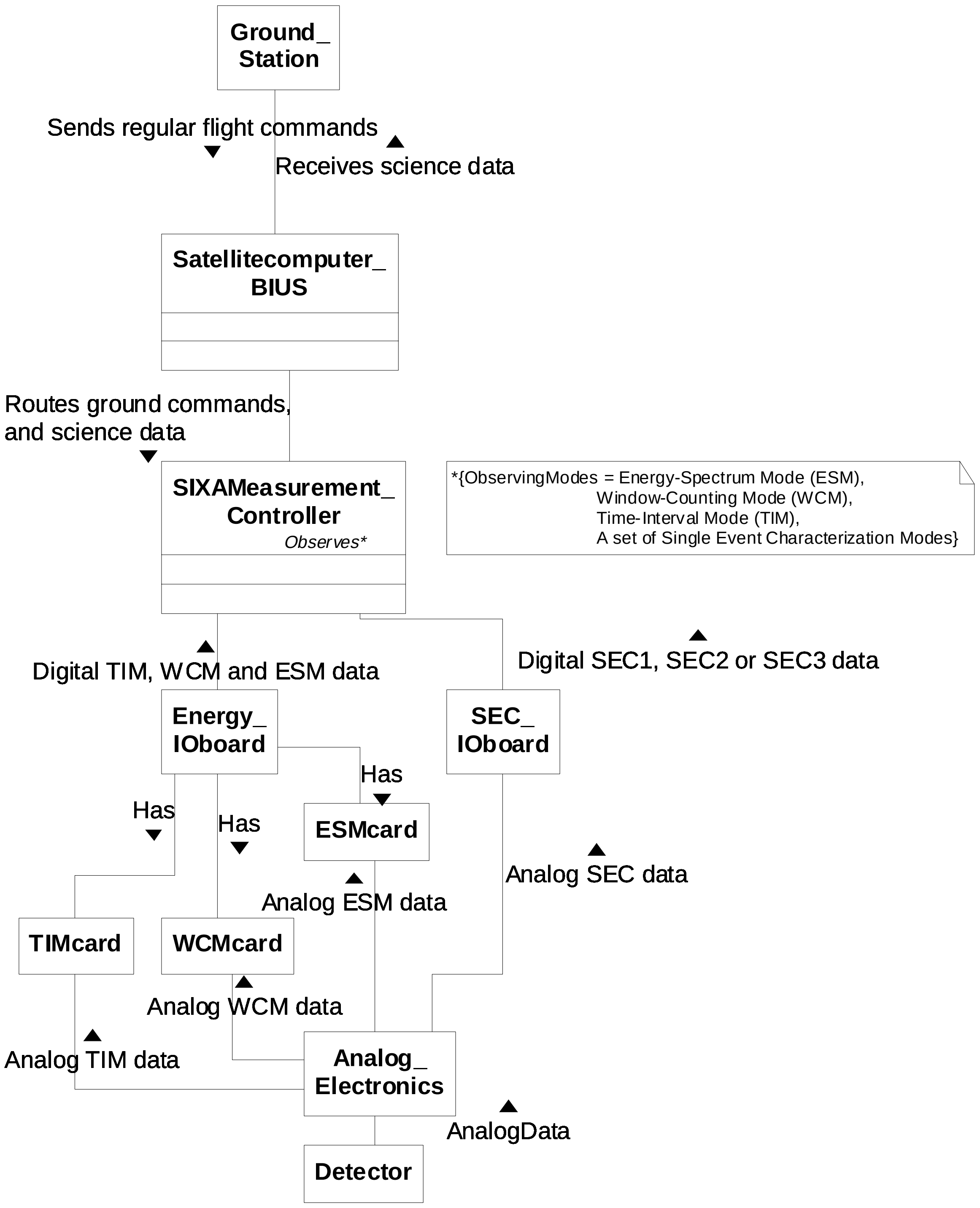

Ground_ Station
Sends regular flight commands
Receives science data
Satellitecomputer_ BIUS
Routes ground commands, and science data
SIXAMeasurement_ Controller
Observes*
*{ObservingModes = Energy-Spectrum Mode (ESM), Window-Counting Mode (WCM), Time-Interval Mode (TIM), A set of Single Event Characterization Modes}
Digital TIM, WCM and ESM data
Digital SEC1, SEC2 or SEC3 data
Energy_ IOboard
SEC_ IOboard
Has
Has
Has
ESMcard
Analog ESM data
Analog SEC data
TIMcard
WCMcard
Analog WCM data
Analog TIM data
Analog_ Electronics
AnalogData
Detector

Figure 3. The system context diagram of the SIXA Measurement Controller.

2.2 System Architecture Definition

The SIXA instrument forms a node in a distributed Spectrum-Roentgen-Gamma mission system [Leppälä et al. 1996]. The architectural design of the system can be illustrated by the Broker architectural pattern [Buschman et al. 1996, Douglass 1998], as shown in Figure 4. The Broker pattern has been used to structure a complex system as a set of decoupled interoperating components. This results in better flexibility, maintainability, changeability and reusability. The Broker pattern includes another pattern that is called the Proxy pattern. The role of a proxy is to decouple clients from their servers.

The ground station plays the role of a client in the Broker pattern, as shown in Figure 4. It sends ground commands to the space craft service system that has the role of the broker in the pattern. Each ground command is given a time stamp, which specifies when it is to be sent to the SIXA Measurement Controller. When the time is reached to execute a command, the service system will give the command to the Satellite Computer BIUS, which serves as a Server Proxy in the pattern. The SIXA Measurement Controller software is decomposed into two independent and almost identical subsystems, called Energy Measurement Controller and SEC Measurement Controller. These function as servers in the Broker pattern. The ground station addresses ground commands to the SIXA Measurement Controller, not to the separated subsystems. The Energy and SEC Measurement Controller subsystems are allocated to independent processors.

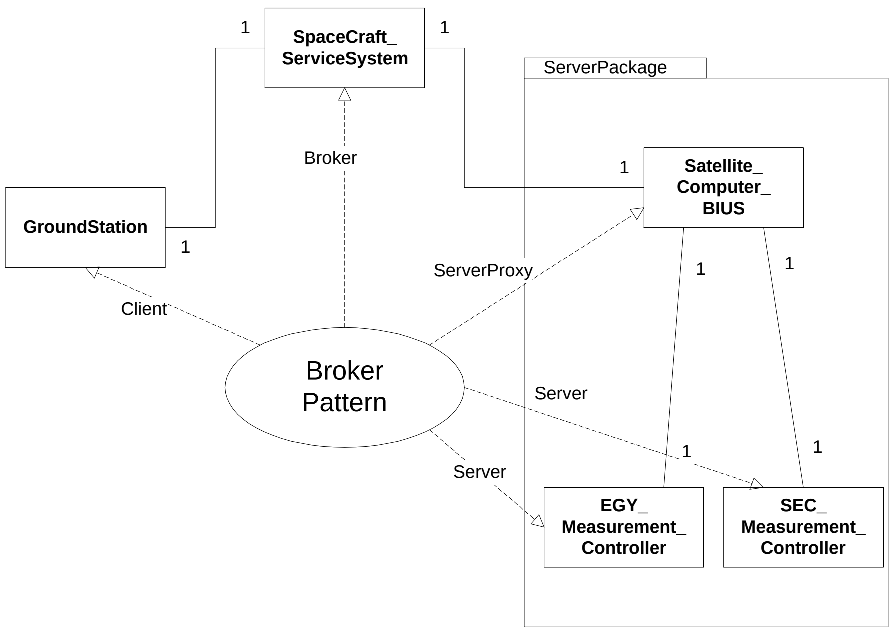

SpaceCraft_ ServiceSystem
1
1
ServerPackage
Broker
1
Satellite_ Computer_ BIUS
GroundStation
1
ServerProxy
1
1
Client
Broker Pattern
Server
1
1
Server
EGY_ Measurement_ Controller
SEC_ Measurement_ Controller

Figure 4. The architecture model of the Spectrum-Roentgen-Gamma mission system from the point of view of the SIXA Measurement Controller.

The role of the Energy Measurement Controller is to control the following three observing modes: Energy-Spectrum Mode (ESM), Window-Counting Mode (WCM) and Time-Interval Mode (TIM). The SEC Measurement Controller, for its part, controls three single event characterisation observing modes.

2.3 Subsystem analysis

The Energy and SEC Measurement Controller subsystems are very similar. The results of a partial analysis of the Energy Measurement subsystem are represented by the following models:

- External Event List,
- Analysis Use Case Models, an MSC scenario is shown in Figure 5, and
- Analysis Object Models.

The external event list defines the external events and their parameters, the sources and arrival patterns of the events, and the expected system response. The Analysis Object Model of the Energy Measurement subsystem is shown in Figure 6. The model comprises the main concepts of the subsystem, the most important properties of the concepts and the relationships between the concepts.

Centralised control architecture is adjusted to the Energy Measurement subsystem (see Figure 6). The architecture is also known as the master-slave architecture. The Measurement Control class is responsible for controlling and timing the main functions of the subsystem. It provides methods for controlling the electronics and instances of the rather passive Energy Data Acquisition and Energy File Management classes.

The Energy Data Acquisition class is responsible for the acquisition of energy data chunks. It provides interfaces for controlling the science data acquisition and includes as well as hides data acquisition details. The Energy File Management class is responsible for the storing of energy data chunks. It provides interfaces for storing data chunks and controlling the transmission of the stored data to the ground station, as well as hiding data storing details.

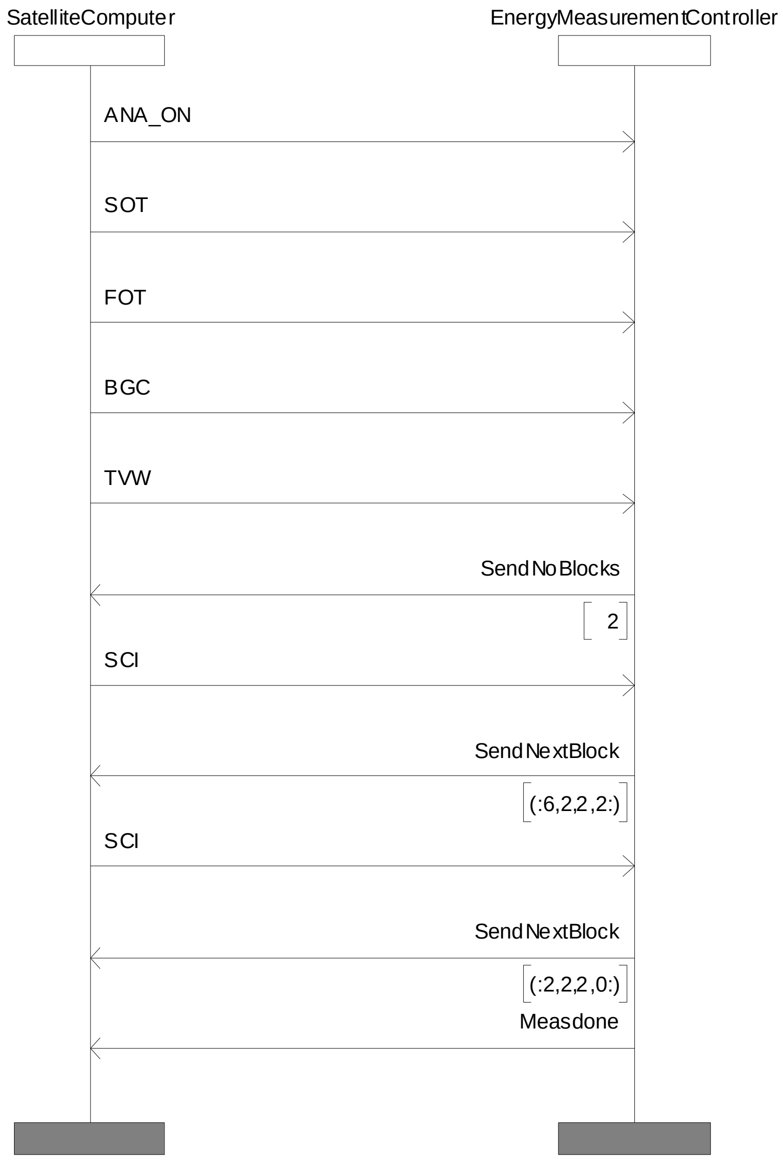


Figure 5. An MSC scenario of the Energy Measurement subsystem.

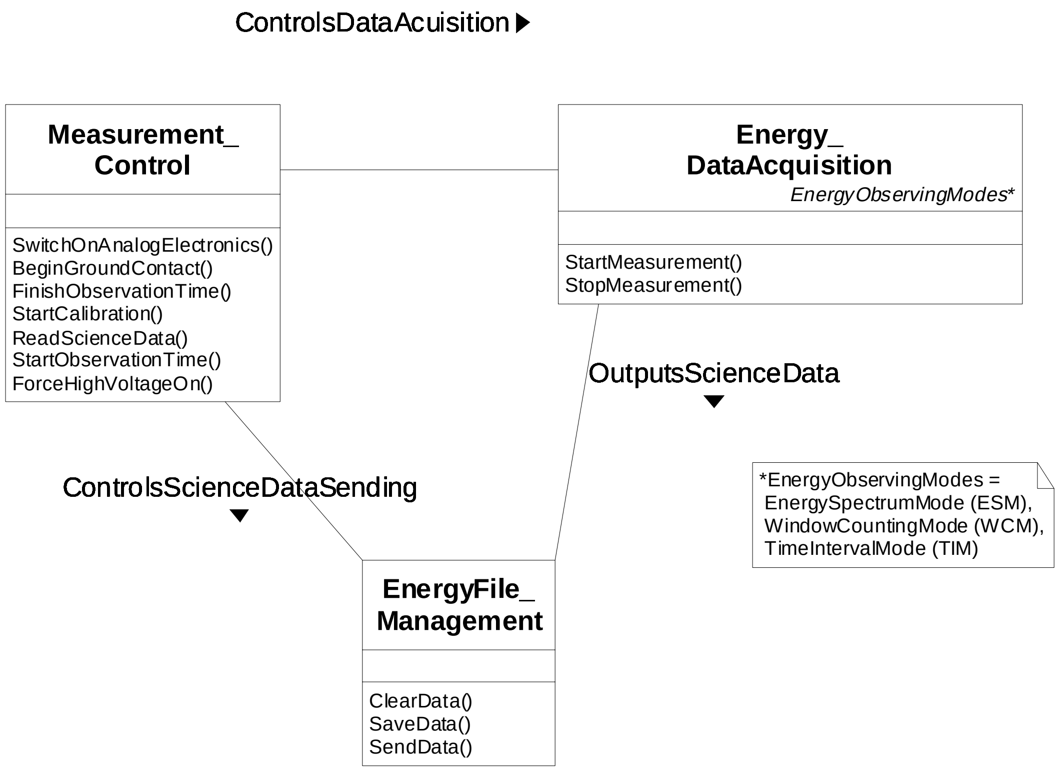


Figure 6. The analysis object model of the Energy Measurement subsystem.

2.4 Subsystem design

In the following paragraphs, the suitability of the SDL and ROOM languages and OORT, SOMT and ROOM methods for the modelling of the Energy Measurement subsystem is treated from various points of view.

2.4.1 Architectural and behavioural design

The mapping of concepts from the analysis object model of the Energy Measurement subsystem into SDL models is very straightforward in this case. The documentation structure of the design models of the Energy Measurement subsystem is shown in Figure 7. The Energy Measurement subsystem is mapped into the EGY Measurement block, the Measurement Control class into the MeasControl process, the Energy Data Acquisition class into the Data Acquisition process and the Energy File Management class into the File Management process. It is also easy to refine the analysis MSC of the Energy Measurement subsystem into a design MSC.

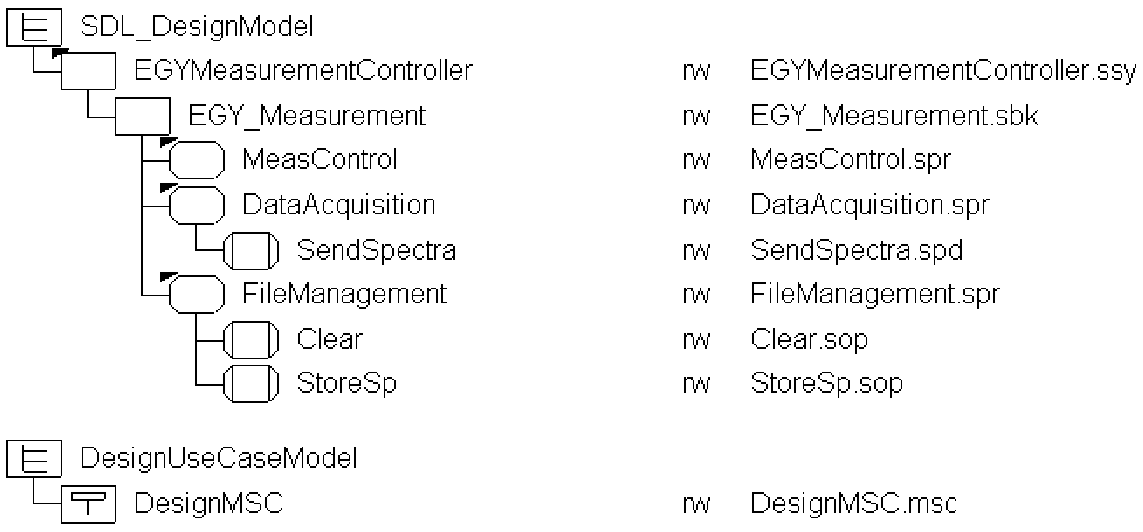

SDL_DesignModel
EGYMeasurementController rw EGYMeasurementController.ssy
EGY_Measurement rw EGY_Measurement.sbk
MeasControl rw MeasControl.spr
DataAcquisition rw DataAcquisition.spr
SendSpectra rw SendSpectra.spd
FileManagement rw FileManagement.spr
Clear rw Clear.sop
StoreSp rw StoreSp.sop
DesignUseCaseModel
DesignMSC rw DesignMSC.msc

Figure 7. The structure of the SDL subsystem design models for the Energy Measurement subsystem.

The block and processes can be identified also in the existing SIXA software, which consists of a few parallel state machines communicating with each other using asynchronous messages. The state machines were modelled using Statecharts [Harel & Rolph 1989] or RT-SA and the connections between the state machines were modelled with RT-SA data flow diagrams. The final code was either hand-written or generated automatically from RT-SA diagrams.

The block diagrams and processes of SDL are semantically quite close to RT-SA semantics used in SIXA modelling. The reverse engineering of SIXA state diagrams and data flow diagrams into corresponding SDL diagrams was quite straightforward. It is not that significant, if the SDL or RT-SA notation is used.

The translation of the SDL diagrams to ROOM diagrams was straightforward. For the Energy Measurement subsystem, the system and block diagrams of the SDL model could be directly reflected upon a corresponding actor structure, ports and bindings in the ROOM language. The hierarchical finite state machine notation of ROOM includes graphic symbols only for states and transitions. Actions, signals and conditions are coded using a detailed coding language, i.e. RPL or C++ in the used ObjecTime version. Therefore, SDL process diagrams would look different from the corresponding ROOM state diagrams, though the translation of the SDL process diagrams to ROOM state diagrams was rather straightforward. A reverse translation may be more difficult, due to the hierarchical structure of the ROOM state diagram.

SDL and ROOM are well suited for the architectural and behavioural modelling of the Energy Measurement subsystem. As a language, SDL is not offering anything remarkably new. However, the tools supporting SDL (especially simulators and validators) and ROOM (simulator) are much more advanced than those of the CASE tool - code generator pair used in the SIXA development. The availability of such a tool would certainly have increased productivity and quality in the design and testing of SIXA.

2.4.2 Data and algorithm modelling

The features of basic SDL for modelling data and corresponding algorithms are rather poor. As the science data structures and algorithms of SIXA are somewhat complex, it would have been really difficult to carry out the modelling with SDL. Furthermore, the data structures of SIXA which are used in external communications (protocol frame structures) are defined at bit level and also the expressing of data format issues is not within the scope of SDL. Therefore, SDL is not suitable for modelling data and algorithms for the SIXA software. The data structures and algorithms may, however, be defined outside SDL with the C language, for example.

Data structures and algorithms are coded using a detailed coding language of ROOM, RPL or C++ in the used ObjecTime version. Only C++ is used to implement the data structures and algorithms of the target software.

2.4.3 Concurrency of processes

The science data collection of SIXA is based on polling the hardware/software interface. The detector electronics insert science data into the hardware FIFO queues, while the software reads data from the FIFOs to data memory. The reading of FIFOs is extremely time-critical. Due to this, the FIFO polling loop is written in assembly and highly speed-optimised by hand.

During observation SIXA stays in the polling loop for several hours. Therefore, the polling loop must be somehow interruptible, so as to allow the other subsystems to operate during data acquisition. A pre-emptive operating system was used in SIXA and the polling loop was assigned to task with relatively low priority, thus allowing for interruptions by other tasks.

The SDL language itself does not contain any task definition properties like the Ada language, for example. The assignment of SDL processes and blocks to tasks must be performed outside the scope of the SDL language.

The need for a pre-emptive operating system is an indication of a broader modelling restriction: SDL is not well-suited for the modelling of continuous processing. With the aid of a pre-emptive scheduler and task concept, it will be easier to model and implement the “continuous” processing of a process. In the SDL model of the Energy Measurement subsystem, the continuos polling loop was simulated with a timer-triggered transition i.e. the continuous polling was replaced with periodic polling. Although this works well in the simulation phase, in practice the polling will have to be continuous in order to meet the requirements set on the data collection speed of SIXA.

The ROOM language supports synchronous and asynchronous communication and message priorities. It is easier to model continuous polling in ROOM than in SDL.

2.4.4 Hardware interface

SDL and ROOM have no means of expressing the features required in hardware interfaces accurately (such as interrupts, specific I/O address space instructions, and the like). These features have to be coded outside the SDL or ROOM model and then imported into SDL or ROOM, or else the hardware interface has to be completely hidden from the SDL or ROOM model, e.g. by separating the hardware-dependent code into hand-written tasks and modules. Simulation models for hardware interfaces may be created using the SDL and ROOM languages.

2.4.5 Testing

The classic way of unit testing is to draw up a test driver which sends input to a module, calls the module and then checks the generated output and internal state of the module after the call. During the testing, the internal activities of the module will not be visible to the test driver; this is called “black box” testing. The use of a test driver enables easy repetition of tests, since test cases and expected results are coded into the test driver.

The use of MSCs as test drivers in the SDL environment has proved a very promising idea. MSCs can be used to generate black box test cases, provided that the signals between the system under test and the environment are charted. The use of MSCs also enables white box and grey box testing, since all or some of the internal events can be included in the test MSC.

The use of coverage tools will further enhance the testing phase. After all the test MSCs have been executed, the coverage tools can be used to check the completeness of the test session.

2.5 Discussion

Domain models and descriptions are used to collect and organise domain knowledge in such a way that will make product development, configuration and production of systems faster and more efficient. However, developing and maintaining domain models and descriptions is not an easy task. In addition, the OORT, SOMT and ROOM methods do not include a domain analysis phase. This feature will be addressed in the following chapter.

The system requirements specification recaptures the relevant product requirements for the software in a way that supports software development. This viewpoint is addressed by the Octopus and Ellison's methods, for example, but not by the OORT, SOMT and ROOM methods. The OORT, SOMT and ROOM methods provide basic guidelines for specifying use cases and scenarios. More sophisticated guidelines are provided, for instance, by the Real-Time UML method. The system context diagram and the external event list are necessary for the SIXA Measurement Controller, but they are not supported by the OORT method. The external event list is not supported by the SOMT method.

The OORT and SOMT methods support OMT for object-oriented analysis, i.e. for understanding the problem domain and analysing it using classes and objects. The methods support only those features of OMT that can easily be mapped into SDL designs. This will probably constrain the UML [UML 1997] support of the methods in the future. UML is the next version of the OMT notation. The UML for Real-Time recommendation combines the UML concepts and the special constructs and formalisms of the ROOM language.

Although object-oriented analysis models are useful, they cannot be regarded as the best foundation for the object-oriented design structure. They do not, for instance, directly convey the benefits of the object-oriented design technology. Good object-oriented design should be based upon the decoupling facilities of polymorphism and encapsulation. This allows object-oriented software to be more flexible, maintainable and reusable. It is difficult to apply object-oriented design patterns in SDL design using the object-oriented extensions in SDL. The OORT and SOMT methods do not provide any guidelines for that.

The OORT, SOMT and ROOM methods pay little, if any, attention to timing constraints and performance analysis, which are among the essential design problems of mission-critical software.

2.6 Summary

Not all the required viewpoints of the system requirements specification, system architecture definition and subsystem analysis of the SIXA Measurement Controller are supported by the OORT, SOMT and ROOM methods. OORT and SOMT support only those features of object-oriented analysis that can easily be mapped into SDL designs, while ROOM includes no notation for object models.

The disadvantages of SDL and ROOM include inadequate support for complex data type, algorithm, timing constraints, performance analysis, concurrency and hardware interface descriptions. These shortcomings restrict the use of SDL and ROOM for system designing and implementation. Some parts of the system will have to be implemented outside SDL or ROOM with the C, C++ or an assembly language. Consequently, the automatic validation process will not work properly.

The interfaces and several object-oriented features of SDL are specific to SDL. The use of object-oriented languages in the implementation of SDL models is not supported. Hence it is difficult to apply sound object-oriented design principles in SDL design. The benefits of the technology of object-oriented design can be only partially achieved.

The SDL and ROOM languages are well suited for the behaviour modelling of the SIXA Measurement Controller. The modelling of state behaviour with the SDL and ROOM state machines and the communication between state machines with block diagrams or actors does not provide any problems. The benefits of the simulation and validation of design models are convincing.

3. Component-oriented development

Most of the object-oriented methods are intended to be used within the context of the traditional water-fall type life-cycle model for forward engineering. In reality, however, embedded systems are rarely developed from scratch by following the ideal waterfall-type life-cycle process. Instead, they are re-engineered by reusing, modifying and extending existing sets of system requirements, specifications, designs and implementations. In general, investing in reusable software structures and components is feasible, if several related control systems are to be developed.

This chapter is concerned with domain analysis, SDL patterns and application frameworks for mission-critical software. Sections 3.1 and 3.2 comprise introductions to domain analysis, SDL patterns and SDL and ROOM frameworks. Section 3.3 will discuss the domain analysis of the X-ray spectrometer control software.

Section 3.4 describes an SDL pattern for designing a centralised control architecture of SDL models for X-ray spectrometer control software. SDL pattern description templates [Geppert & Rössler 1996] are utilised to describe the pattern. The presented examples are derived from the results of the domain analysis in Section 3.3 and from the results of the work that was described in Chapter 2.

The SDL pattern has been used in designing the control architecture of SDL models in the SDL framework for the measurement subsystem of X-ray spectrometer controllers, which will be described in Section 3.5. The SDL pattern described in Section 3.4 has been used in designing the control architecture of SDL models in the framework. A corresponding ROOM framework has been developed and will be discussed later. The utilisation of general design patterns in the construction and reuse of the frameworks will also be covered.

The experience gained in the development of the SDL pattern and the frameworks is will be discussed in Section 3.6. CASE tool support and the applicability of the object-oriented features of SDL in the construction of the SDL framework will also be addressed. Finally, Section 3.7 provides a summary of the chapter.

3.1 Introduction to domain analysis

Component-oriented development should rely on carefully analysed domains. Domain models and descriptions are used to collect and organise domain knowledge in a way that will make product development, configuration and production of systems faster and more effective. The are a number of definitions for the terms domain and domain analysis. The definitions of the TIMe [Braek & Haugen 1993] and FODA [Hess et al. 1990] methods are applied in this document.

The result of the TIMe domain analysis is represented by two models: a domain object model and a domain property model. Interaction properties in domain property models are described by means of MSC. The domain property model may also include textual property descriptions. Textual domain statement and concept dictionary descriptions are recommended.

There are three phases in the FODA domain analysis process:

- Context Analysis. The resulting models are structural and context diagrams. There is no definition for the notation of the structure diagram. The context diagram is a top-level data flow diagram.
- Domain Modelling. The results are entity relationship models, features models, functional models (Statecharts and Activity charts [Harel & Rolph 1989]) and a domain terminology dictionary.
- Architecture modelling. The results are process interaction models and module structure charts.

The domain concept in TIMe covers common phenomena, concepts and processes that are not specific to any particular system family, but rather to a market segment. The domain concept of TIMe is more restricted than that of FODA. The ESA standard PSS-05-0 [ESA 1991] and the OORT, SOMT and ROOM methods do not include any domain analysis activity.

3.2 Introduction to SDL patterns and frameworks

Most SDL-based life cycle approaches such as the OORT [OORT 1996] and SOMT [SOMT 1996] methods support developing independent application systems. They generally consider the possibility of model reuse between systems very little. However, it often makes sense to create and acquire a set of reusable components, design patterns and frameworks, if several related control systems are to be developed. The object orientation of SDL provides a promising framework for making explicit reusable system structures and software components. Yet, the organised reuse of SDL specifications and patterns is not a very common phenomenon. A summary of the SDL pattern and framework development described in this chapter has been published in [Ihme 1998b].

An SDL pattern is defined in [Geppert et al. 1998] as “a reusable software artifact representing a generic solution for a recurring design problem with SDL as the applied design language”. The approach of Geppert and Rössler [Geppert & Rössler 1996] has been applied and adapted in the development of the SDL pattern that will be described in this chapter.

An SDL-specific framework is defined by the TIMe method [Braek & Haugen 1993] as follows: “A class or family of systems, with predefined structure so that a specific system only has to provide the specific contents of part of this structure." A framework may be defined also as other kinds of types, e.g. as a procedure type. Actual systems or components may be described by defining subtypes and redefining the application-specific types in the framework. The same definition is fit for ROOM frameworks, as well.

SDL packages may be used in the implementation of SDL frameworks. These packages are libraries that are used for making reusable SDL components available in different systems. The reusable components are defined as SDL types and fragments.

3.3 The spectrometer domain example

Domain analysis defines the features and capabilities of a class of related systems. Domain models define the structures, functions, objects, classes, and relationships in a domain. The spectrometer software domain is a set of current and future spectrometers that share a set of common software features and system features that are closely related to software.

The results of a partial analysis of the spectrometer domain are represented by the following models:

- *Object Models*, the general concepts of the domain
    - *Domain Context Diagram*, general interaction relationships between systems and external actors in the domain
    - *Domain Object Model*, the general concepts of spectrometer measurement features
    - *Product Family Model,* the classification of the spectrometer products.
- *Domain Analysis Use Case Models*, typical interaction sequences between systems and external actors in the domain

Special attention has to be paid to identifying the scope and limits of the domain that will be re-engineered for reuse. Although artifacts in the domain should offer a notable potential for reusability, often there are no statistics available for the occurrence frequency of software parts. The domain should be coherent, meaningful and not too large. Although a single component is not often reusable on its own, it allows reusing in conjunction with many other related components.

The measurement subsystem of a spectrometer control software was selected as the domain. An example subsystem is the SIXA Measurement Controller described in Chapter 2. The measurement domain included the spectrometer operating modes Measurement, Data Retention and Dumping. Hardware interfaces and data file content were largely excluded from the domain.

The spectrometer was going to be installed in a satellite and controlled by a ground station via a satellite as shown in the Domain Context Diagram in Figure 8. An array detector was going to be used for detecting X-ray photons.

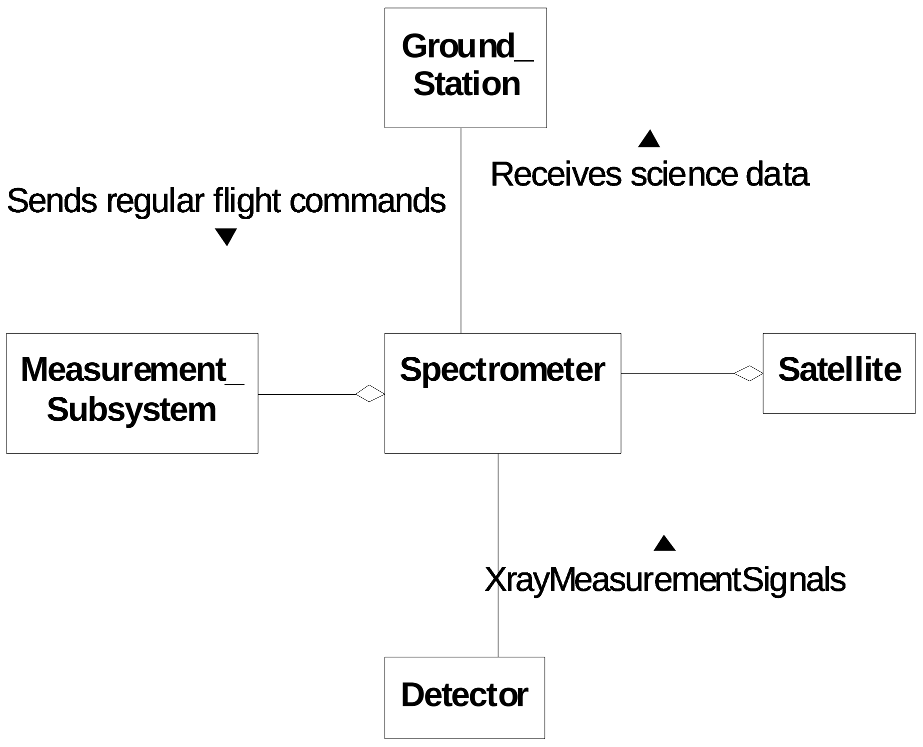


Figure 8. The context diagram of the measurement domain.

The Domain Object Model in Figure 9 depicts the main concepts in the measurement domain and the relationships between the concepts.

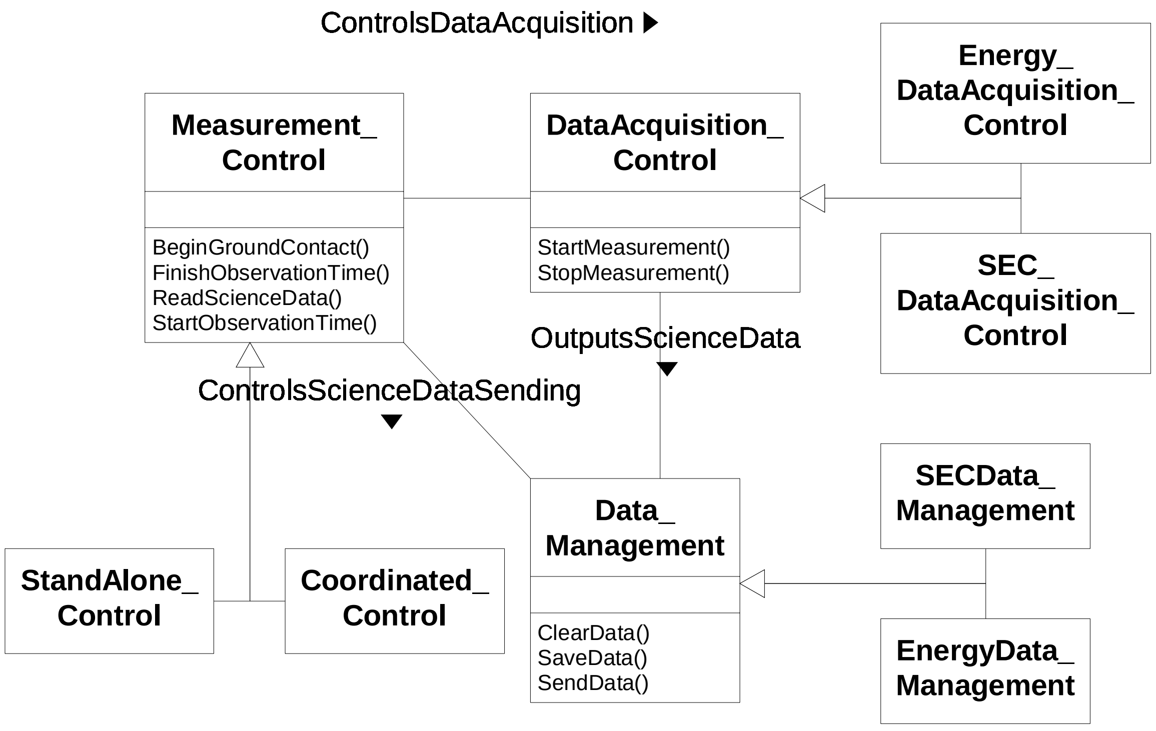


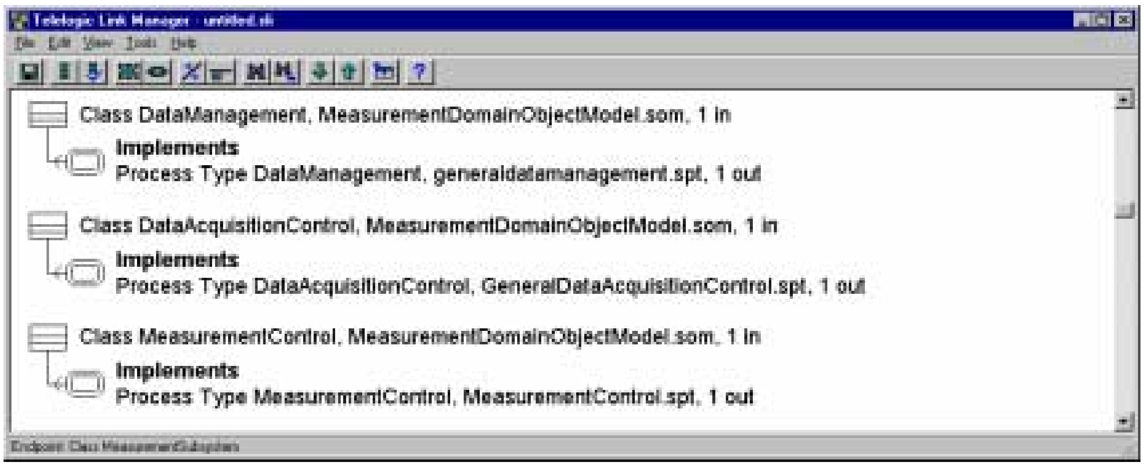
Figure 9. The object model of the measurement domain.

The centralised control architecture was discussed in Chapter 2. The architecture was adjusted to the measurement domain, see Figure 9. The Measurement Control class is responsible for controlling and timing the main functions of the system. It provides methods for controlling instances of the rather passive Data Acquisition Control and Data Management classes.

There are two subtypes to the Measurement Control class: Stand Alone Control and Coordinated Control. The Stand Alone Control class provides methods for controlling the analog electronics of stand alone spectrometers. The Coordinated Control class provides methods for controlling analog electronics on behalf of several spectrometers.

The Data Acquisition Control class provides interfaces for controlling science data acquisition and includes as well as hides data acquisition details. The Data Management class provides interfaces for storing science data and controlling the transmission of the stored data to the ground station, as well as hiding data storing details.

There are two subtypes to the Data Acquisition Control class: Energy Data Acquisition Control and SEC (Single Event Characterization) Data Acquisition Control. Correspondingly, there are also two subtypes to the Data Management class: Energy Data Management and SEC Data Management. Implements links from the SDL models of the measurement subsystem to the classes in the Measurement Domain Object Model are shown in Figure 10.


Telelogic Link Manager - untitled.sli
File Edit View Tools Help

Class DataManagement, MeasurementDomainObjectModel.som, 1 in
Implements
Process Type DataManagement, generaldatamanagement.spt, 1 out

Class DataAcquisitionControl, MeasurementDomainObjectModel.som, 1 in
Implements
Process Type DataAcquisitionControl, GeneralDataAcquisitionControl.spt, 1 out

Class MeasurementControl, MeasurementDomainObjectModel.som, 1 in
Implements
Process Type MeasurementControl, MeasurementControl.spt, 1 out

Endpoint: Class MeasurementSubsystem

Figure 10. Implements links from SDL processes to the classes in the Measurement Domain Object Model.

A typical scenario of an activity in embedded control software includes the following sequences: initialize activity, start activity, stop activity, and send activity data. The scenarios of the measurement subsystem include these kinds of sequences. Sequence diagrams in Figures 11 and 12 depict a typical use sequence of the measurement subsystem:

- Repeat for each target:
    - Switch on analog electronics
    - Receive target coordinates
    - Start observation time
    - Finish observation time
    - Switch off analog electronics
- Begin ground contact
- Transmit science data

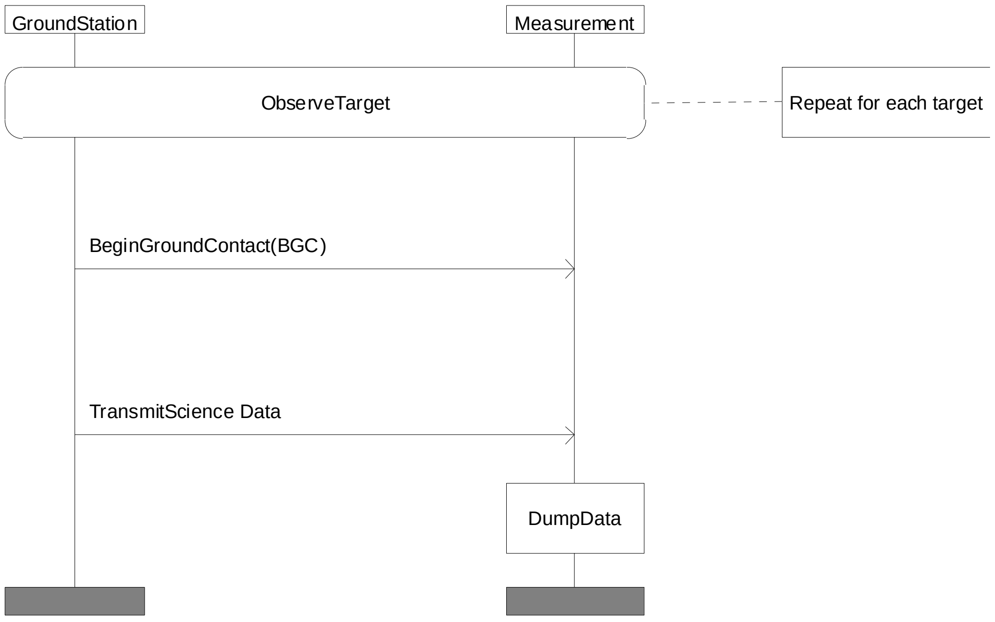


Figure 11. A typical observation sequence use case of the measurement subsystem.

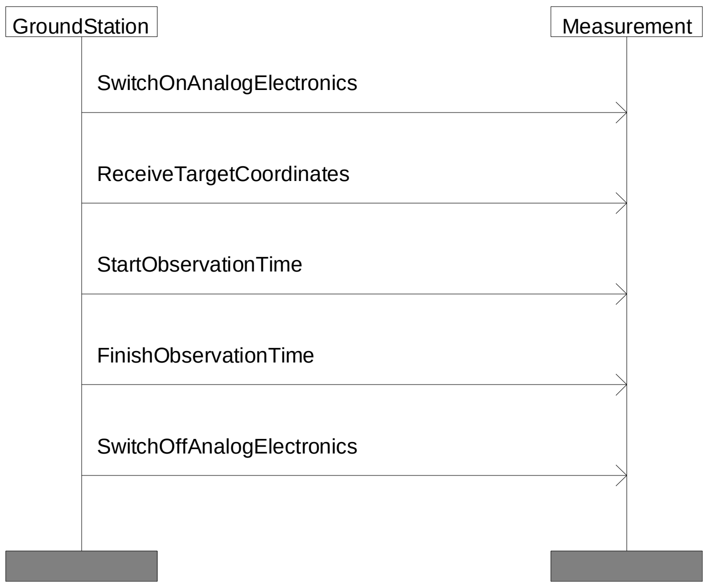


Figure 12. The Observe Target use case model of the measurement subsystem.

An example of the product family of spectrometer controllers is shown in the Product Family Model in Figure 13. Energy Controller and SEC Controller are the base product types in the product category Spectrometer Controller. The SEC Controller products control the single event characterization observing modes. The EGY Controller products control three observing modes: Energy-Spectrum Mode (ESM), Window-Counting Mode (WCM), and Time-Interval Mode (TIM). The following classes represent concrete products in the product family: EGY Controller, Autonomous SEC Controller and SECwithEGY Controller.

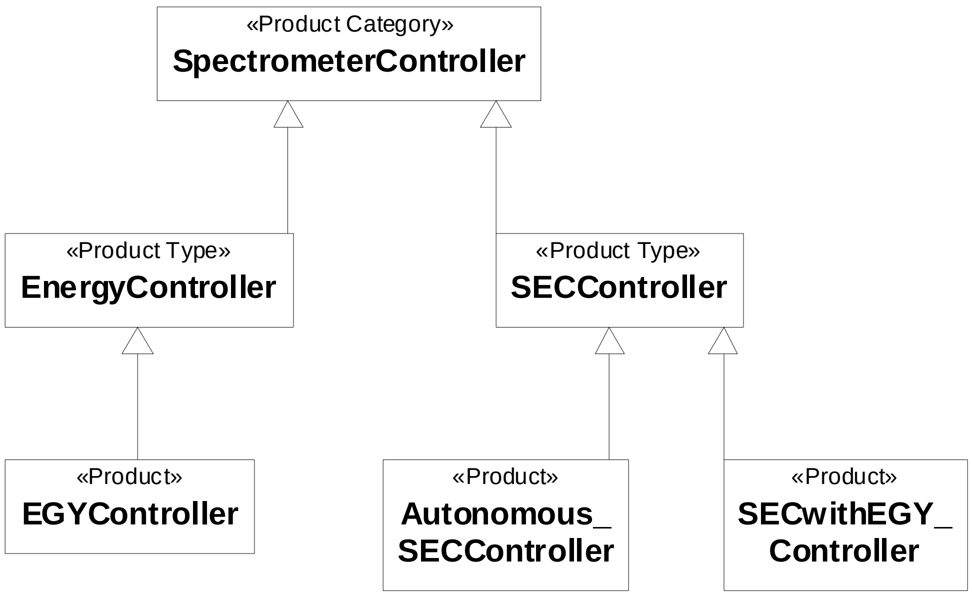
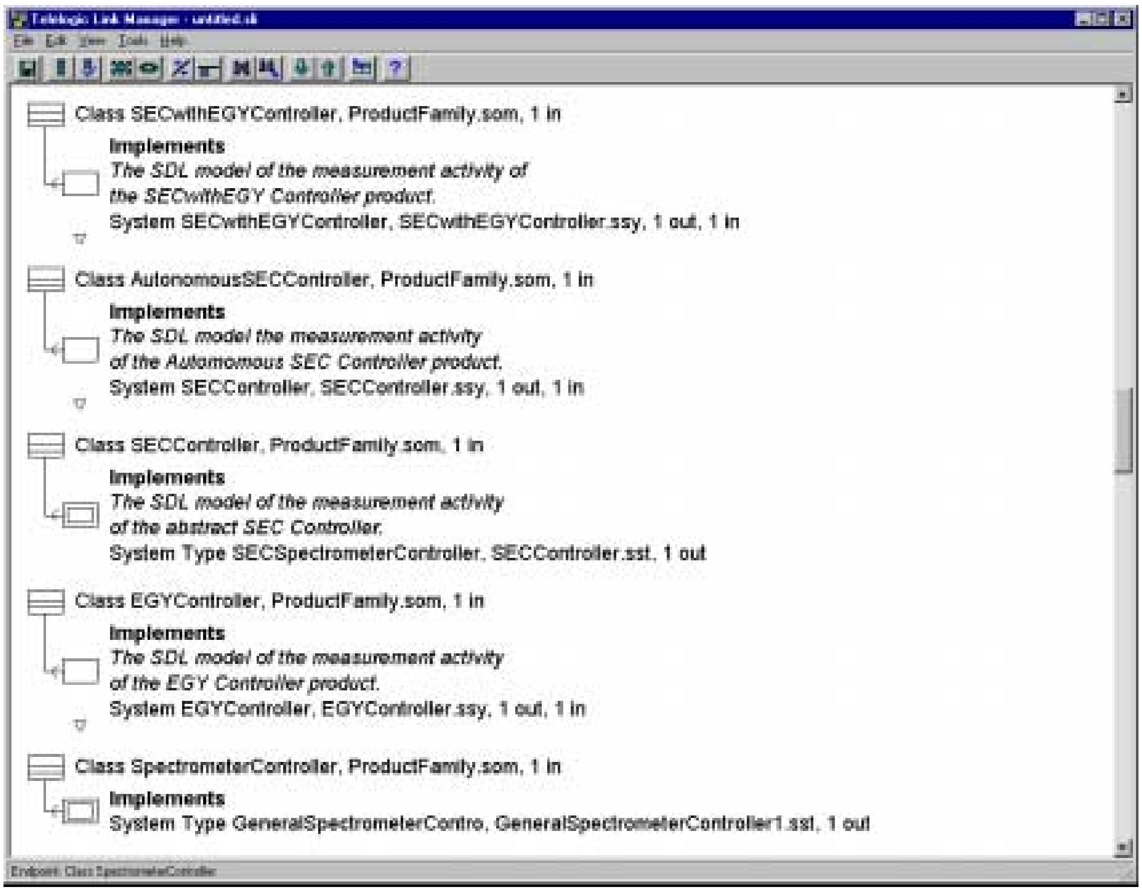


Figure 13. The product family of spectrometer controllers.

The SEC DPU of the SIXA instrument is an instance of the SECwithEGY Controller class. The EGY DPU of the SIXA instrument is an instance of the EGY Controller class. Implements links from the SDL models of the measurement subsystem to the classes in the Product Family Model are shown in Figure 14.


Telelogic Link Manager - untitled.slk
File Edit View Tools Help

Class SECwithEGYController, ProductFamily.som, 1 in
Implements
The SDL model of the measurement activity of the SECwithEGY Controller product.
System SECwithEGYController, SECwithEGYController.ssy, 1 out, 1 in

Class AutonomousSECController, ProductFamily.som, 1 in
Implements
The SDL model the measurement activity of the Automomous SEC Controller product.
System SECController, SECController.ssy, 1 out, 1 in

Class SECController, ProductFamily.som, 1 in
Implements
The SDL model of the measurement activity of the abstract SEC Controller.
System Type SECSpectrometerController, SECController.sst, 1 out

Class EGYController, ProductFamily.som, 1 in
Implements
The SDL model of the measurement activity of the EGY Controller product.
System EGYController, EGYController.ssy, 1 out, 1 in

Class SpectrometerController, ProductFamily.som, 1 in
Implements
System Type GeneralSpectrometerContro, GeneralSpectrometerController1.sst, 1 out

Endpoint: Class SpectrometerController

Figure 14. Implements links from SDL models to the classes in the Product Family Model.

3.4 The Measurement Control architectural pattern

3.4.1 Intent

The architectural pattern of Measurement Control introduces a centralised control architecture for the measurement subsystem of X-ray spectrometer controllers. The structure, main concepts and relationships between the concepts in the Measurement Control pattern are depicted in the OMT object model in Figure 15, derived from the Measurement Domain object model in Figure 9. The MSC diagram in Figure 16 depicts a typical message scenario for the measurement subsystem. The diagram is derived from the MSC scenario of the Energy Measurement subsystem described in Chapter 2.

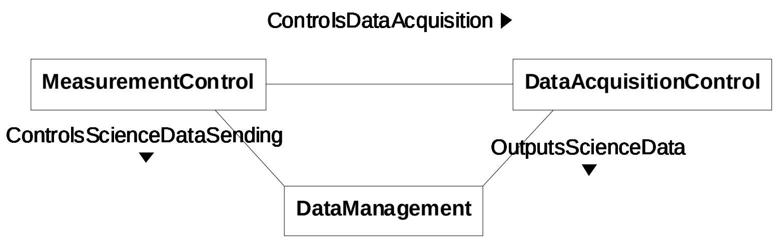


Figure 15. The structure of the measurement control architecture.

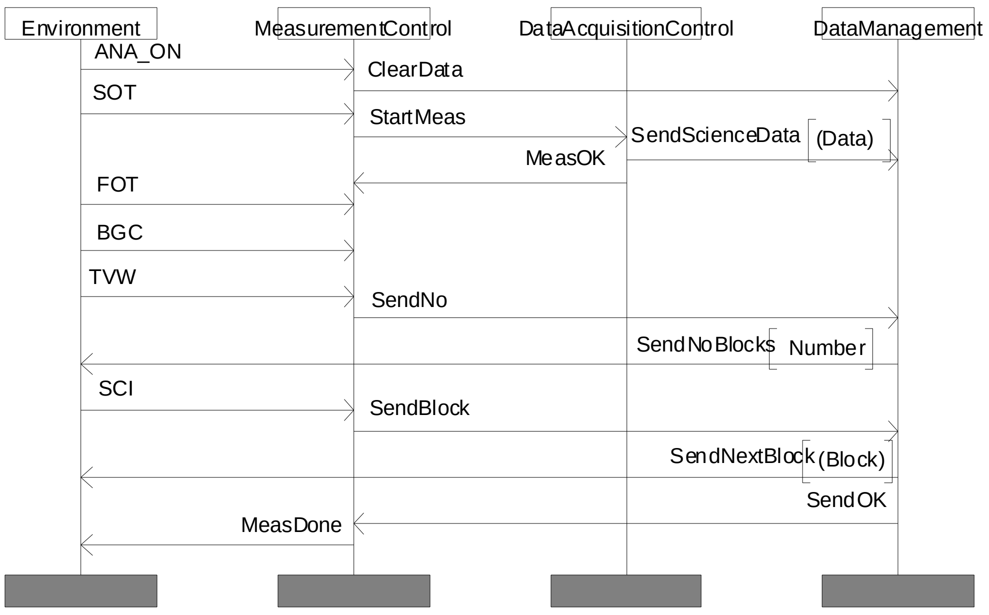


Figure 16. A message scenario typical of the measurement subsystem.

3.4.2 Motivation

Centralised control architecture is very common in the embedded control software of various products, as well as in industrial equipment and scientific instruments. The architecture is also known as master-slave architecture. The complexity of control is centralised on the master. This makes it easy to modify and maintain the software, provided that the system does not get too complex when the distributed control architecture becomes simpler. The master-slave architecture is well suited for hard real-time systems requiring complete timing predictability.

3.4.3 Participants in the pattern

The Measurement Control class is responsible for controlling and timing the main functions of the subsystem. It provides methods for controlling the electronics and instances of rather the passive Data Acquisition Control and Data Management classes. The Data Acquisition Control class is responsible for the acquisition of energy data chunks. It provides interfaces for controlling the science data acquisition and includes as well as hides data acquisition details. The Data Management class is responsible for the storing of energy data chunks. It provides interfaces for storing data chunks and controlling the transmission of the stored data to the ground station, as well as hiding data storing details.

In SDL models, the Measurement Control class is mapped into the Measurement Control process, the Data Acquisition Control class into the Data Acquisition Control process and the Data Management class into the Data Management process. The Data Acquisition Control and Data Management components are described in the following sections. The Measurement Control component is too complex to be described here.

3.4.4 Data Acquisition Control

The SDL fragment of Data Acquisition Control is shown in Figure 17. The Data Acquisition Control component introduces basic data acquisition services for the Measurement Control component. After initiating a data acquisition activity by sending the StartMeas signal, Measurement Control waits for a MeasOK signal from Data Acquisition Control. The signal indicates that the data has been acquired. Measurement Control may stop a data acquisition activity by sending the StopMeas signal that is replied by Data Acquisition Control using the MeasOK signal. The states DA_WAIT_START and DA_WAIT_STOP may be renamed.

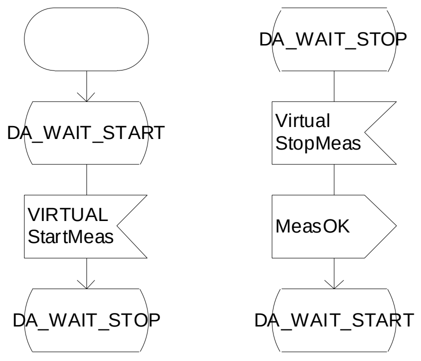


Figure 17. The SDL fragment of Data Acquisition Control.

Semantic properties of the Data Acquisition Control SDL fragment

If the assumptions stated below hold, Measurement Control will eventually receive the MeasOK signal from Data Acquisition Control after sending the StartMeas or StopMeas signal. The assumptions are as follows:

- The StartMeas, StopMeas and MeasOK signals are not implicitly consumed by the superclasses.
- The transmission of the signals between Measurement Control and Data Acquisition Control is reliable.
- The MeasOK signal will always be sent before the state DA_WAIT_START.
- The state DA_WAIT_START will eventually always be reached.

Redefinition of the Data Acquisition Control SDL fragment

The embedded SDL fragment will be supplemented by additional statements for acquiring spectrum data from an array detector and sending spectrum signals to Data Management. The above property still holds, if the pattern is redefined by the introduction of additional statements, which do not disrupt or bypass the thread of control from predefined input to predefined output statements. However, the thread of control from the StopMeas input signal will be bypassed by threads of control that end in the MeasOK output statement and the state DA_WAIT_START.

During observation in the DA_WAIT_STOP state, the thread of control stays in the polling loop of the hardware/software interface for several hours. The polling will have to be continuous to meet the requirements set on the data collection speed. For the simulation purposes of the system, continuous polling will be replaced with periodic polling using a timer-triggered transition from the DA_WAIT_STOP state.

3.4.5 Data Management

The SDL fragment of the Data Management component is shown in Figure 18. The Data Management component introduces basic data management services for the Measurement Control component. The acquired science data is saved as data blocks by Data Management. Measurement Control controls Data Management by the following signals:

- ClearData: The science data is cleared.
- SendNo: The signal SendNoBlocks with the number of blocks is sent to the environment.

- SendBlock: The signal SendNextBlock with a data block is sent to the environment. If all blocks have been sent, then the signal SendOK is sent to the Measurement Control.

Transitions from the FM_WAIT state will normally be added by the designer for saving the science data associated with the spectrum signals from Data Acquisition Control. The state FM_WAIT may be renamed.

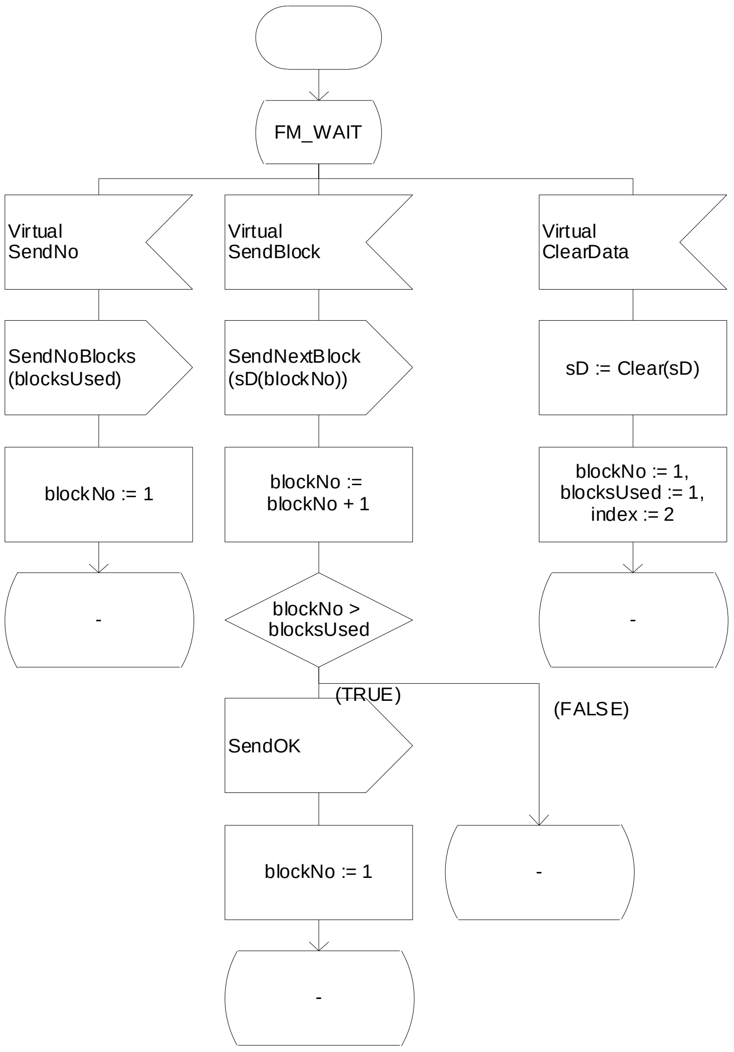


Figure 18. The SDL fragment of Data Management.

Semantic properties of the Data Management SDL fragment

The SendNoBlocks signal has to be sent to the environment with the number of blocks after the SendNo signal from Measurement Control. The SendNextBlock signal has to be sent to the environment with the next data block after the SendBlock signal from Measurement Control.

Property 1: If the assumptions stated below hold, then Measurement Control will eventually receive the SendOK signal from Data Management after sending the SendBlock signal. The assumptions are as follows:

- The ClearData, SendNo and SendBlock signals are not implicitly consumed by the superclasses.
- The indexes and counters in Data Management are properly initialised and modified.
- The state FM_WAIT will eventually always be reached.

Redefinition of the Data Management SDL fragment

The embedded SDL fragment will be supplemented by additional transitions and statements for saving the science data sent by Data Acquisition Control. Property 2 determines the allowed redefinitions:

- Property 2: Property 1 still holds, if the pattern is redefined by the introduction of additional transitions from the FM_WAIT state for saving the science data associated with spectrum signals from Data Acquisition Control. The indexes and counters in Data Management must be properly initialised and modified in the additional transitions. The state FM_WAIT must eventually always be reached at the end of the additional transitions.

3.5 SDL and ROOM frameworks for the spectrometer controller family

This section describes the SDL and ROOM frameworks that were developed for a family of spectrometer controllers. The SDL framework includes reusable SDL components and the corresponding ROOM framework ROOM components.

The Measurement Control pattern described in Section 3.3 is used in designing the control architecture of SDL models in the SDL framework. The domain models described in Section 3.2 as well as general design patterns [Gamma et al. 1995, Douglass 1998] are used in designing the structure of the SDL and ROOM frameworks as well as the architecture of the framework components.

3.5.1 Interfaces for spectrometer controllers

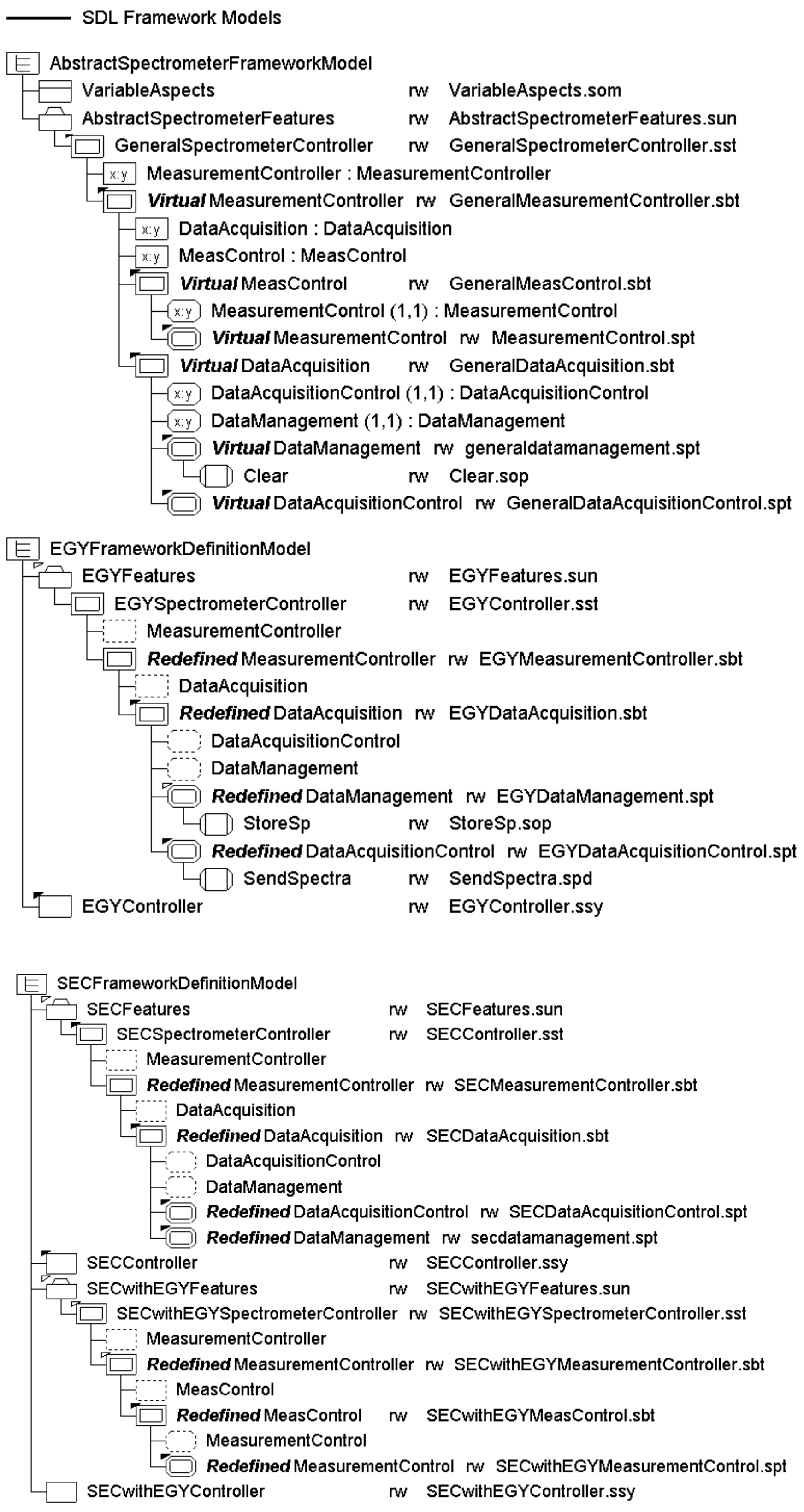
The documentation structure of the SDL framework is shown in Figure 19. The SDL models are partitioned in three modules of the Telelogic SDT tool: Abstract

Spectrometer Framework Model, EGY Framework Definition Model and SEC Framework Definition Model.

The SDL types in the Abstract Spectrometer Features package, in Figure 19, define common structures and interfaces for the measurement subsystems of spectrometer controllers. The SDL types are not completely defined for simulation and there are also no instances of them. A signal type cannot be virtual and redefined in SDL. The signals from the Data Acquisition Control process to the Data Management process are usually of different types. Therefore, the signals are defined in concrete SDL models and not in the Abstract Spectrometer Features package.

The interfaces of the General Spectrometer Controller system type and all virtual block and process types in the Abstract Spectrometer Features package are inherited as such into the SDL models of the three different spectrometer controller types, which differ in terms of how they operate internally. This makes use of the Strategy Pattern [Gamma et al. 1995] that describes selections of different implementations of one kind of black box behaviour. The pattern is also known as Policy [Douglass 1998]. The purpose of the strategies is to simplify interfaces, improve reuse, highlight potential variability, and help deciding upon different concrete components in the framework. SDL system, block and process types have been used to abstract strategies from the documentation of existing spectrometer controller software. These serve as reusable units in the framework.

The General Spectrometer Controller system type, the MeasControl block and the Data Acquisition block define the most important interfaces in the SDL Framework. The General Spectrometer Controller system type hides the differences between concrete controllers. The MeasControl block hides control details from other parts of the system. The Data Acquisition block hides differences of data acquisition components from other parts of the system. The content of the packages in the SDL framework is included in [Ihme 1998c].

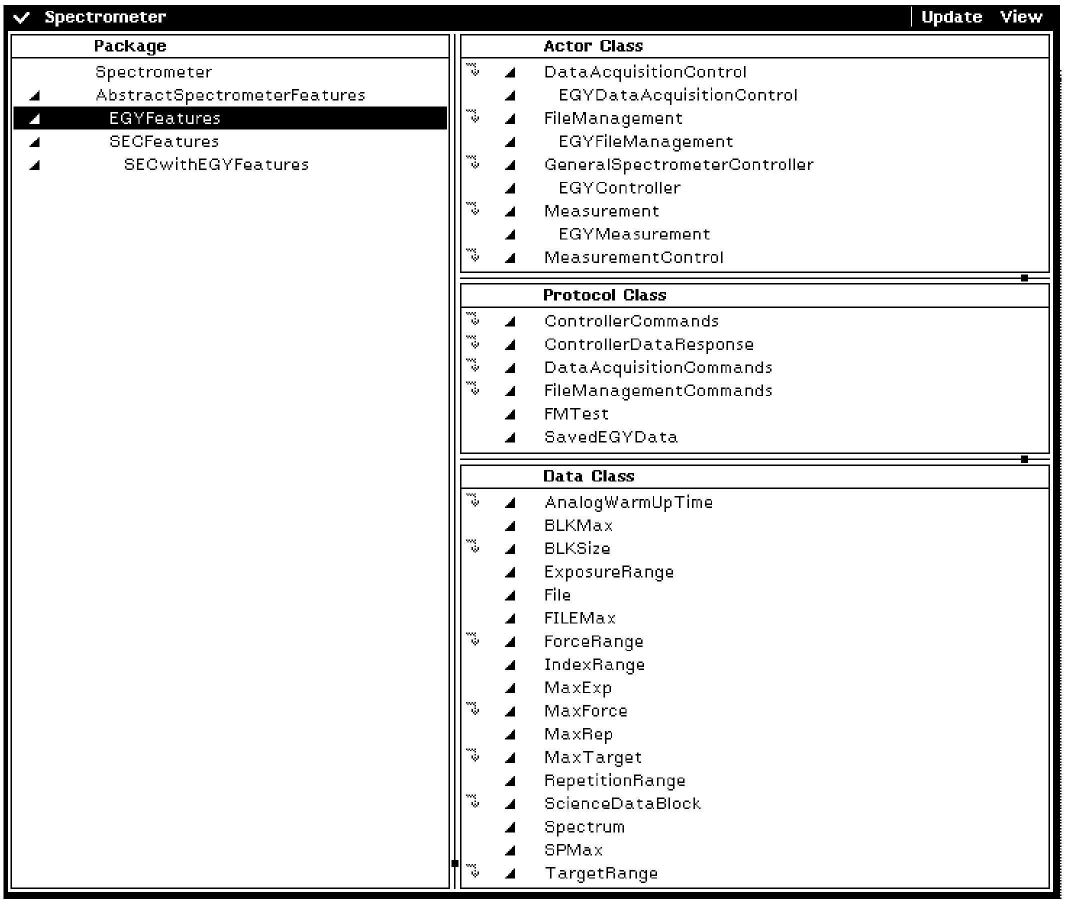


```
SDL Framework Models

AbstractSpectrometerFrameworkModel
  VariableAspects                              rw   VariableAspects.som
  AbstractSpectrometerFeatures                 rw   AbstractSpectrometerFeatures.sun
    GeneralSpectrometerController              rw   GeneralSpectrometerController.sst
      MeasurementController : MeasurementController
      Virtual MeasurementController  rw   GeneralMeasurementController.sbt
        DataAcquisition : DataAcquisition
        MeasControl : MeasControl
        Virtual MeasControl          rw   GeneralMeasControl.sbt
          MeasurementControl (1,1) : MeasurementControl
          Virtual MeasurementControl  rw   MeasurementControl.spt
        Virtual DataAcquisition      rw   GeneralDataAcquisition.sbt
          DataAcquisitionControl (1,1) : DataAcquisitionControl
          DataManagement (1,1) : DataManagement
          Virtual DataManagement  rw   generaldatamanagement.spt
            Clear                    rw   Clear.sop
          Virtual DataAcquisitionControl  rw   GeneralDataAcquisitionControl.spt

EGYFrameworkDefinitionModel
  EGYFeatures                                  rw   EGYFeatures.sun
    EGYSpectrometerController                  rw   EGYController.sst
      MeasurementController
      Redefined MeasurementController  rw   EGYMeasurementController.sbt
        DataAcquisition
        Redefined DataAcquisition  rw   EGYDataAcquisition.sbt
          DataAcquisitionControl
          DataManagement
          Redefined DataManagement  rw   EGYDataManagement.spt
            StoreSp                  rw   StoreSp.sop
          Redefined DataAcquisitionControl  rw   EGYDataAcquisitionControl.spt
            SendSpectra              rw   SendSpectra.spd
  EGYController                                rw   EGYController.ssy


SECFrameworkDefinitionModel
  SECFeatures                                  rw   SECFeatures.sun
    SECSpectrometerController                  rw   SECController.sst
      MeasurementController
      Redefined MeasurementController  rw   SECMeasurementController.sbt
        DataAcquisition
        Redefined DataAcquisition  rw   SECDataAcquisition.sbt
          DataAcquisitionControl
          DataManagement
          Redefined DataAcquisitionControl  rw   SECDataAcquisitionControl.spt
          Redefined DataManagement  rw   secdatamanagement.spt
  SECController                                rw   SECController.ssy
  SECwithEGYFeatures                           rw   SECwithEGYFeatures.sun
    SECwithEGYSpectrometerController  rw   SECwithEGYSpectrometerController.sst
      MeasurementController
      Redefined MeasurementController  rw   SECwithEGYMeasurementController.sbt
        MeasControl
        Redefined MeasControl        rw   SECwithEGYMeasControl.sbt
          MeasurementControl
          Redefined MeasurementControl  rw   SECwithEGYMeasurementControl.spt
  SECwithEGYController                         rw   SECwithEGYController.ssy
```

Figure 19. The documentation structure of the spectrometer controller SDL framework.

3.5.2 A ROOM framework for the spectrometer controller family

A snapshot of the spectrometer controller ROOM framework is shown in Figure 20. It is taken from the Model Browser window of the ObjecTime tool. The Model Browser supports the navigation through all packages and classes in the model. The same packages as in the package list in Figure 20 are shown in the SDL framework in Figure 19. The partitioning of spectrometer controller models into packages is similar in both frameworks. The increased indent of the EGY and SEC Features packages shows that they are the child packages of the Abstract Spectrometer Features package. The first item in the packages list, the Spectrometer, is a special entry: it is the update itself, not a package.



```
Package                              Actor Class
  Spectrometer                       ↘ ◢ DataAcquisitionControl
◢ AbstractSpectrometerFeatures         ◢   EGYDataAcquisitionControl
◢   EGYFeatures                      ↘ ◢ FileManagement
◢   SECFeatures                        ◢   EGYFileManagement
◢     SECwithEGYFeatures             ↘ ◢ GeneralSpectrometerController
                                       ◢   EGYController
                                     ↘ ◢ Measurement
                                       ◢   EGYMeasurement
                                     ↘ ◢ MeasurementControl

                                     Protocol Class
                                     ↘ ◢ ControllerCommands
                                     ↘ ◢ ControllerDataResponse
                                     ↘ ◢ DataAcquisitionCommands
                                     ↘ ◢ FileManagementCommands
                                       ◢ FMTest
                                       ◢ SavedEGYData

                                     Data Class
                                     ↘ ◢ AnalogWarmUpTime
                                       ◢ BLKMax
                                     ↘ ◢ BLKSize
                                       ◢ ExposureRange
                                       ◢ File
                                       ◢ FILEMax
                                     ↘ ◢ ForceRange
                                       ◢ IndexRange
                                       ◢ MaxExp
                                     ↘ ◢ MaxForce
                                       ◢ MaxRep
                                     ↘ ◢ MaxTarget
                                       ◢ RepetitionRange
                                     ↘ ◢ ScienceDataBlock
                                       ◢ Spectrum
                                       ◢ SPMax
                                     ↘ ◢ TargetRange
```

Figure 20. A snapshot of the spectrometer controller ROOM framework.

The actor, protocol and data class lists in Figure 20 show the classes contained in the currently selected EGY Features package. Arrows at the beginning of the rows denote classes inherited from the Abstract Spectrometer Features package.

The functionality of the classes in the ROOM framework corresponds to the functionality of the SDL framework. ObjecTime allows manipulating collections of objects and inheritance relationships of classes in hierarchical ROOM models. It supports defining and navigating inheritance relationships between classes in models located in different packages. A detailed description of the ROOM framework is presented in [Ihme 1998c].

3.5.3 Reusability of the SDL framework

The SDL framework supports several strategies for reusing SDL components, e.g.

- The selection of concrete SDL system models that have the same type of interface to the Ground Station (Figure 21).
- The selection of an existing system type for designing a new version of the measurement subsystem. For example, the SDL model in the SECwithEGY Features package in Figure 19 is designed by inheriting and redefining the SEC Spectrometer Controller system type in the SEC Features package.
- The selection of the Abstract Spectrometer framework model for designing a new framework model. For example, the EGY and SEC framework definition models are designed by inheriting and redefining definitions in the Abstract Spectrometer Features package.

The last two selections are followed by several lower level selections, such as

- the selection of the control components of the spectrometer software that have the same type of interface to the Ground Station (Figure 22), and
- the selection of data acquisition components having the same type of interface to the Ground Station (Figure 23).

The designer can use the strategy pattern for selecting different Spectrometer Controller system models as shown in Figure 21. The Ground Station (Client) uses the same type of interface for each controller. The interface is defined by the General Spectrometer Controller system type (Abstract Strategy). General Spectrometer Controller subclasses implement three different concrete SDL system models: EGY Controller, SEC Controller and SECwithEGY Controller. This means that the observation command sequences sent by the Ground Station remain unchanged regardless of which concrete controller instance of the General Spectrometer Controller type is installed or active in the satellite.

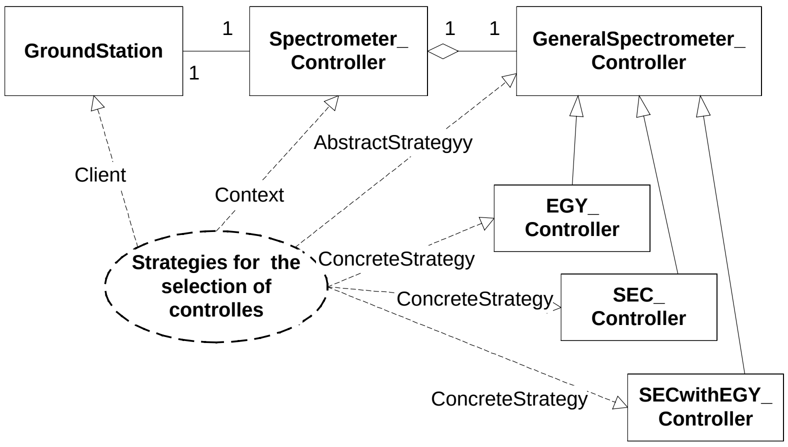


Figure 21. The use of the strategy pattern for the selection of Spectrometer Controller system models.

The MeasControl block type defines the interface of the Measurement Control process type that is responsible for controlling and timing the main functions of the system. MeasControl subclasses implement two different concrete control strategies (Figure 22):

- StandAloneControl implements the control of analog electronics for stand alone spectrometers. The EGY and SEC Controllers have the StandAloneControl variation of the Measurement Control process.
- CoordinatedControl implements the control of analog electronics on behalf of several spectrometers. The SECwithEGY Controller has the CoordinatedControl variation of the Measurement Control process. Only one spectrometer in a set of spectrometers is recommended to have the CoordinatedControl variation.

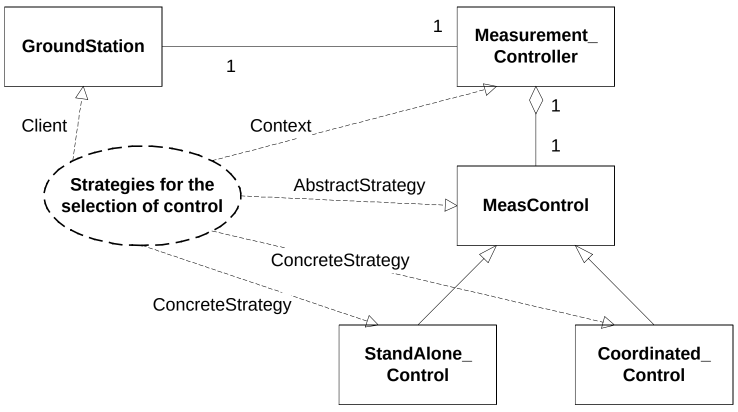


Figure 22. The use of the strategy pattern for the selection of concrete control processes.

The Ground Station uses the same type of interface for each spectrometer measurement controller regardless of the observing modes of the controllers, as shown in Figure 23. The abstract Data Acquisition block type defines interfaces for the aggregate of the Data Acquisition Control and Data Management processes. The Data Acquisition subclasses implement two different concrete measurement strategies:

- EGYDataAcquisition implements the control of three EGY observing modes: Energy-Spectrum Mode, Window-Counting Mode and Time-Interval Mode.
- SECDataAcquisition implements the control of three single event characterisation observing modes.

The strategies in Figures 22 and 23 let control or measurement algorithms vary independently of the Ground Station (client) command protocol. However, the Ground Station must be aware of the used measurement strategy because of the varying content of the science data. In addition, Data Acquisition and MeasControl subclasses can vary independently from each other as long as the properties defined by the Measurement Control pattern are not violated.

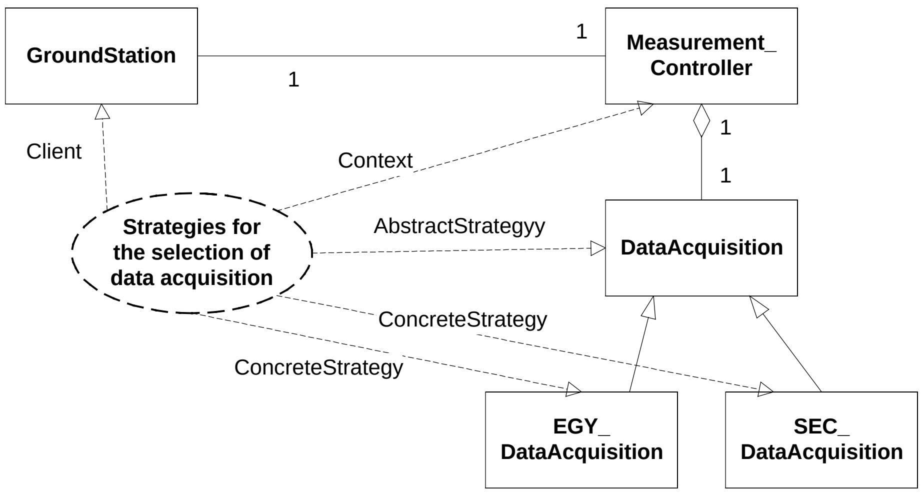


Figure 23. The use of the strategy pattern for the selection of concrete data acquisition strategies.

This discussion about the reusability of the SDL framework is fit for the ROOM framework and will not be repeated here.

3.5.4 Documentation of the framework variability

To maximise the usefulness and reusability of SDL framework components, these should be able to offer some degree of variability. To allow reusing, these small, flexible and general components often need to be specialised in some way.

A new SDL type may be based on a supertype exported from the SDL framework through the object-oriented concepts of inheritance and specialisation. Through the specification of the subtype, not only can new properties be added, but also properties of the supertype can be redefined. Structure, behaviour and data are additive SDL properties, while only structure and behaviour can be redefined. Adding a new property when inheriting a supertype is an example of implicit variability. A structural or behavioural SDL type (not system type) or a transition can be defined as being virtual by adding the keyword 'virtual' in front of the type specification or the signal name. This kind of explicit variability is carefully controlled. The ROOM language does not support controlled modification of inherited characteristics. It is very important to document the designed implicit variability of components and collections of components. The documentation of explicit variability will help the designer to find and understand all virtual definitions incorporated in the SDL framework.

The Abstract Spectrometer Features package serves as a solid platform for various Spectrometer Controllers. The various concrete SDL system models in the SDL framework and in the Spectrometer Controller domain differ with regard to some variable aspects. The implemented variable aspects and variability points of the SDL framework are associated with the reuse strategies. Most new variable aspects are estimated to be associated with the strategies.

The variable aspects of the General Spectrometer Controller in the SDL framework are documented in the Variable Aspects OMT model, see Figure 24. A similar model may be created for the General Spectrometer Controller in the ROOM framework. The VariationPoint links of the aspect objects in the model are used to identify the locations in the SDL framework at which variations will occur.

3.5.5 Dependence between component systems in the frameworks

The spectrometer controller SDL framework can be seen as a kind of component system. The system comprises a set of component subsystems that are the feature packages in the framework. The structure of the component system should be flexible, maintainable, reusable and robust. The feature packages may be used as releasable units and they can be assigned to individuals or teams for implementation, maintenance and support.

Figure 25 shows the dependencies between the packages in the spectrometer controller SDL framework. There is no notation for packages in the OMT object model editor of the SDT tool. Therefore, the packages are represented by class symbols, see Figure 25. There are two kinds of packages in the figure: SDL component systems and actual systems. Stereotypes are used for differentiating between the package types. The actual system packages are SDL systems in this example.

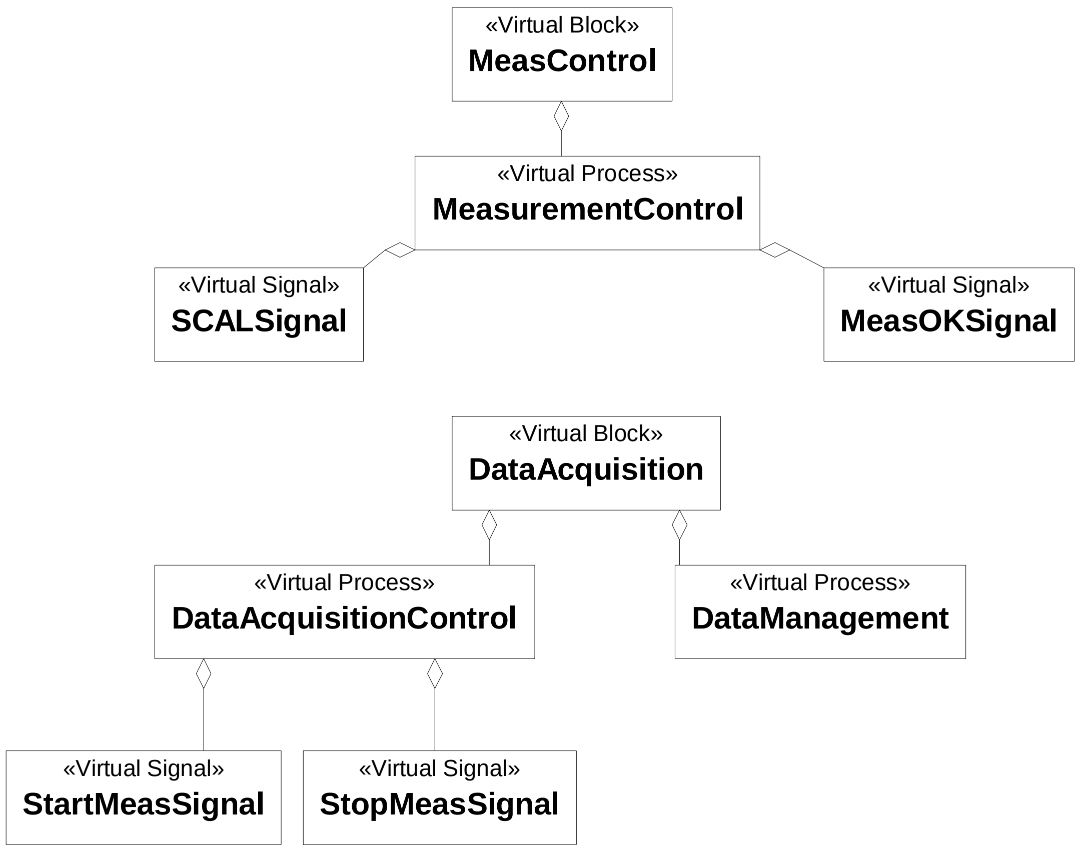


Figure 24. Variable aspects of the General Spectrometer Controller.

The packages depend upon each other, because there may be use or inheritance relationships between the components in different packages. The dependency relationships between the packages point downwards. The structure of this diagram is directed and acyclic. The dependency structure of packages proceeds in a bottom-up fashion. The dependency is highest at the top of the diagram and lowest at the bottom. The packages at the top depend heavily upon other packages, while none depends upon them. The package at the bottom is not dependent on other packages.

The EGY Controller, SEC Controller and SECwithEGY Controller packages are specific for each Spectrometer system. No other package depends upon them. The SECwithEGY Features package depends upon the SEC Features package. In addition, it depends upon the Abstract Spectrometer Features package indirectly.

The Measurement Control process in the SECwithEGY Controller package has to be changed, for example, if the duration of the EGY and SEC measurements is changing. This change has no impact on the other packages. The SEC Features package is likely to be changed, if the single event characterization modes are changing. These changes will affect the SEC Controller, SECwithEGY Features and SECwithEGY Controller packages. The AbstractSpectrometerFeatures package is not dependent on any other package, whereas all the other packages depend upon it, either directly or indirectly.

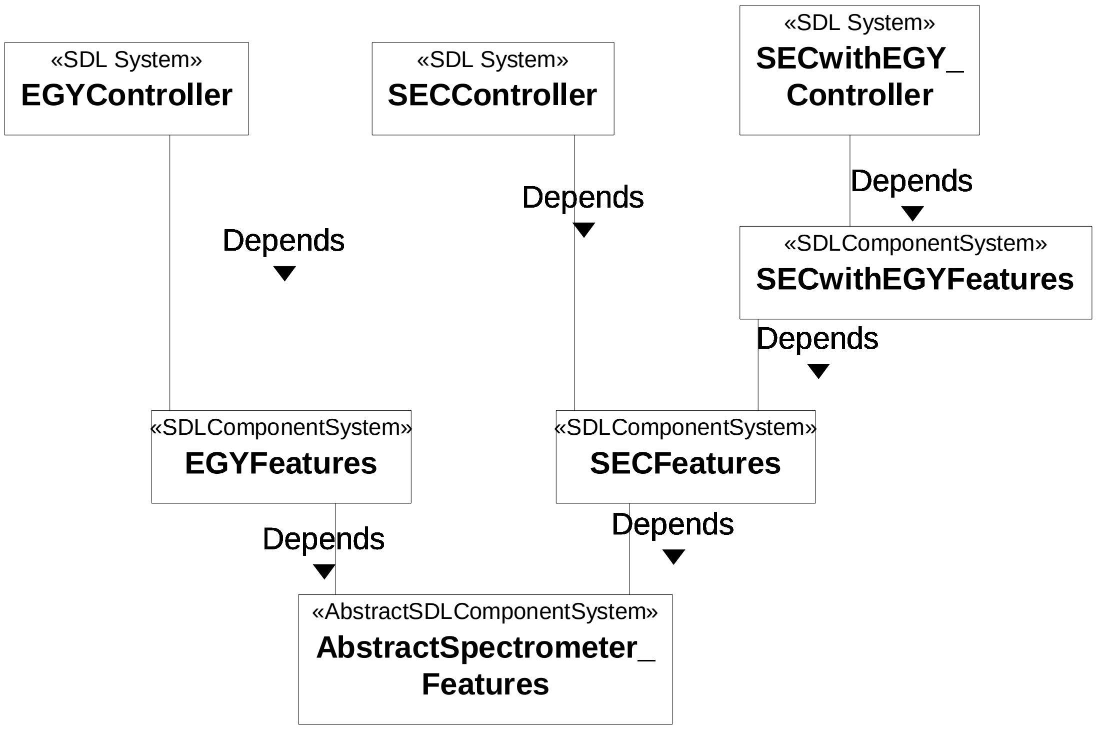


Figure 25. Dependencies between the packages in the spectrometer controller SDL framework.

3.6 Discussion

Domain models and descriptions may make product development, as well as systems configuration and production of systems, faster and more effective. However, it is not easy to develop and maintain the domain models and descriptions. Domain models are useful particularly for the development of reusable components and component systems and to novices in specific application domains. On the other hand, experts may consider proposed domain models very simple and self-evident.

The Measurement Domain object model in Figure 9 bears a strong resemblance to a TIMe domain object model. It is also very much like an object-oriented version of the

FODA entity relationship model. The Typical Observation Sequence model in Figures 11 and 12 is like a part of the domain property model in TIMe. The Domain Context Diagram in Figure 8 is an object-oriented version of the FODA context diagram.

The OMT structure model and message scenarios of the proposed Measurement Control pattern were derived from the results of a domain analysis, from the Measurement Domain object model and the Typical Observation Sequence model. The OMT and MSC notations were used in the description of the structural aspects and typical interactions of the Measurement Control pattern as proposed by Geppert and Rössler [Geppert & Rössler 1996].

Scenarios for the products in the spectrometer controller product family are intentionally similar. One of the development goals was that the Ground Station would use a single type of interface for the various controllers. In real life, however, volatile scenarios of interfaces are often a problem in the development of mission-critical software. The 1996 version of the MSC specification supports repetitive MSC fragments. The used CASE tool versions did not support MSC 96. An approach for adaptive scenarios [Ihme 1997] was proposed in the Space2000/sw project.

The approach of Geppert and Rössler [Geppert & Rössler 1996] focuses on SDL process patterns in communication protocols. The proposed Measurement Control pattern is a master-slave architectural pattern for the SDL models of spectrometer controllers. The roles of the components in the Measurement Control pattern differ from the component roles in the communication protocol patterns. It proved sensible to include a separate description of the properties of each component in the description of the Measurement Control pattern. The components in the SDL patterns of Geppert and Rössler [Geppert & Rössler 1996] do not have any separate descriptions.

The textual property description proposed by Geppert and Rössler [Geppert & Rössler 1996] proved to be adequate for the rather passive slave components of the Measurement Control pattern, but inadequate for the master component involving complex control sequences and timing constraints. The use of textual descriptions as well as MSCs for the definition of pattern properties may be useful. The interactional properties are described in the domain property models of TIMe using MSCs and textual descriptions. MSC property models may be used to validate redefined SDL patterns using SDL simulator tools. The industry is more interested in MSCs than textual definitions.

SDL patterns may be used for a formal definition of the interfaces of components that are to be partly implemented by means other than SDL. However, the SDL interface support reveals some shortcomings. The communication through gates and channels characterises the current SDL support for interfaces. The interfaces and several object-oriented features of SDL are specific to SDL. The use of object-oriented languages in the implementation of SDL models is not supported. Therefore, it is difficult to apply the already existing sound object-oriented design principles and patterns in SDL design. The decoupling facilities of polymorphism and encapsulation provide a good basis for any object-oriented design. This allows object-oriented software to be more flexible, maintainable and reusable. Thus, the benefits of the technology of object-oriented design can be only partially achieved.

There is ongoing work on standardising an SDL interface definition language that will be based on general object-oriented interface languages. The language will provide a better support for interfaces and will also permit the use of non-SDL components.

The proposed SDL framework of spectrometer controllers includes not only SDL processes, but also complete SDL models for measurement subsystems. The SDL types at different hierarchy levels were planned to be reused. It proved difficult to develop, understand and maintain the framework by means of the ObjectGEODE and SDT methodologies. These methodologies failed to provide an adequate support for the object-oriented features of SDL and for navigating between models in different packages. Only textual inheritance definitions were provided. The error messages in inheritance definitions were often useless and misleading. The user interface of ObjecTime performed much better in the development of the corresponding ROOM framework, allowing to manipulate collections of objects and inheritance relationships of classes in hierarchical ROOM models. It provides a better support for defining and navigating inheritance relationships between classes in models located in different packages.

Special means are needed for specifying and applying application-specific configuration rules for the components in the SDL framework. For example, only one spectrometer controller in a set of controllers can be allowed to have the CoordinatedControl variation of the MeasControl block type.

3.7 Summary

The models proposed for the measurement domain of spectrometer controllers proved very useful for the development of SDL patterns as well as SDL and ROOM frameworks. Examples and experiences gained in applying SDL patterns and frameworks to the designing of embedded control software for spectrometer controllers were presented in thisreport. An existing SDL pattern description approach and templates were applied and adapted to the development of a master-slave architectural pattern, called the Measurement Control pattern. The SDL pattern approach and templates proved well suited to the rather passive slave components of the spectrometer software. However, problems were encountered in the description of complex master components. Not all the required architectural viewpoints of the Measurement Control pattern were supported by the templates. The use of MSCs for the definition of pattern properties in addition to textual descriptions was proposed.

The proposed Measurement Control pattern had an important role in designing the architecture of SDL models in the SDL framework for a family of spectrometer controllers. The existing strategy pattern appeared to be useful in the documentation of component interfaces and design, as well as that of the reuse strategies of the framework. The following viewpoints and problems were encountered during the development of the frameworks:

- Not all components of spectrometer controller software can be implemented in SDL or ROOM. The framework should allow attaching non-SDL or non-ROOM components.
- SDL lacks a sound interface support.
- Special means are needed for the configuration rules of components in the frameworks.
- Dedicated means are needed for the documentation of variability and dependencies between component systems in the frameworks.
- The SDL framework proved difficult to develop, understand and maintain by means of the existing CASE tools.

The ObjectGEODE and SDT tools failed to provide an adequate user interface support for the object-oriented features of SDL and for manipulating and navigating between models in different packages. The user interface of ObjecTime proved much more suitable for the development of the corresponding ROOM framework. The differences between the tools derive partly from the differences between the SDL-92 and ROOM languages.

4. Evaluation of functional aspects of real-time methodologies

Modern real-time software methodologies are sophisticated software development environments consisting of an integrated set of different methods, notations and tools. Methods, notations and tools cannot be evaluated and selected independently due to the integration. Functional aspects are the most important properties of a methodology. This chapter describes the evaluation of the functional aspects of real-time methodologies using the ObjectGEODE methodology as an example. The evaluation of other aspects is discussed in the next chapter.

The first section of this chapter provides an introduction to the performed evaluation. The second section describes the structure and usage of the evaluation framework. The results of the evaluation of the ObjectGEODE tool support for the OORT method are presented in the third section. The suitability of the ObjectGEODE methodology with the ESA software engineering standards is described in the fourth section, and the suitability with the application-specific needs in the fifth section. Finally, the sixth section summarises the outcome of this chapter.

4.1 Introduction to the evaluation of functional aspects

The evaluation of functional aspect should be started by determining which software engineering methods are to be adopted, then proceeding to evaluating and selecting the tools to find the ones best suited to the needs of the organisation. This is often a very complicated and confusing process. Modern real-time methodologies, such as ObjectGEODE, are sophisticated software development environments, consisting of an integrated set of different methods, notations and tools. The frameworks proposed for the assessment of real-time software development methods, such as [Wood et al. 1988], seldom address the integrated methodologies. The frameworks are usually too general and laborious for small companies developing real-time mission-critical software.

This chapter presents a summary of the evaluation of the functional aspects of the ObjectGEODE methodology that consists of the OORT method [OORT 1996] and the ObjectGEODE tool (versions 1.0.1 and 1.1) by Verilog SA. The methodology was evaluated in three main phases:

1. An understanding about the functionality of the selected methodology was gained in the first phase. The tool support of the proposed method (the OORT method) was also evaluated.

2. The suitability of the methodology with the general software engineering standards, in this case [ESA 1991], applied by the organisation, was evaluated in the second phase.

3. Finally, the evaluation concentrated on the requirements and needs specific to the applications to be developed.

The Results of this evaluation have also been presented in [Kumara et al. 1998], [Suihkonen 1998], [Suihkonen & Kumara 1997a] and [Suihkonen & Kumara 1997b].

4.2 The evaluation framework

4.2.1 The structure of the evaluation criteria

The used hierarchical evaluation framework included the following three high level criteria:

- Support for the OORT method
- ESA standards
- Application requirements

Each high level criterion included a set of low-level criteria as shown in Table 1. Each low level criterion included a set of evaluation questions. The number of the criteria was dependent on the logical structuring and number of the questions. The evaluator specified a numerical credit for each question. A scale number defined a maximum numerical credit for each question.

A maximum score was defined for each low level criterion. The score of a low level criterion was calculated using the following formula:

$$\text{Score}_L = \text{MaximumScore}_L * (\Sigma \text{ CreditsOfQuestions}) / (\Sigma \text{ ScalesOfQuestions})$$

The score of each high level criterion was the total of the scores given to the associated low level criteria.

The organisation could make its own estimate of the relative importance of each high and low level criterion or question (weighting). The maximum scores of the high and low level criteria were thus modifiable. Similarly, the importance of each question could be lowered or raised by modifying the scale number.

4.2.2 Capability profiles of the methodology

The average capability profiles yielded by three separate evaluations of the ObjectGEODE methodology are shown in Table 1. The same maximum scores were used in each evaluation. The profiles of the low level criteria in Table 1 are expressed as a percentage of the given score to the maximum score of each low level criterion:

$$\text{Profile}_L = (\text{Score}_L / \text{MaximumScore}_L) * 100\ \%$$

The profiles of the high level criteria are expressed as a percentage of the total of the given scores to the total of the maximum scores of the low level criteria:

$$\text{Profile}_H = (\Sigma\ \text{Scores}_L) / (\Sigma\ \text{MaximumScores}_L) * 100\ \%$$

Table 1. Average capability profiles of the ObjectGEODE methodology as percentages of maximum scores.

	Percentage
Support of the OORT method	**64 %**
Support for Object Modelling	54 %
Support for Use Case Modelling	58 %
Support for Coherence between Object Model and Use Case Model	39 %
Support for Relationships between RA and other Development Activities	64 %
Support for SDL	75 %
Support for Relationships between Architectural Design and Detailed Design	63 %
Support for Relationships between Architectural Design and Test Design	67 %
Detailed Design	66 %
Test Design	67 %
Targeting	67 %
Testing	67 %
OORT Iterative Process	58 %
Methodology Evolution	70 %
ESA Standards	**58 %**
Product Standards	64 %
Process Standards	53 %
Application Requirements	**65 %**
Application Requirements	65 %

The percentages of the capability profiles are converted to the following four textual credits of the criteria using the following rules:

Textual Credit	Percentage
Not satisfied	0 – 39.9
Partially satisfied	40 - 59.9
Largely satisfied	60 – 79.9
Fully satisfied	80 - 100

The textual credits are defined as follows:

- "Not satisfied" indicates that the methodology does not have this function or property or the methodology only very weakly supports the function or property and that the criterion is not satisfied.
- "Partially satisfied" indicates that the methodology weakly supports the function or property in question and the criterion is partially satisfied.
- "Largely satisfied" indicates that the methodology significantly supports this function or property and the criterion is largely satisfied.
- "Fully satisfied" indicates that the methodology completely or almost completely supports this function or property and the criterion is fully satisfied.

The textual credits are used in the following section for interpreting the numerical capability profiles. The precise numerical profiles are presented in Table 1.

4.3 Support for the OORT method

The support of the OORT method criterion helps to estimate how well the ObjectGEODE tool automates and supports the OORT method in the different phases of a software engineering process. The criterion is largely satisfied. The associated low-level criteria in each development phase are discussed in the following sections.

The major activities of the OORT in the model construction and their links are depicted in Figure 26. The requirement analysis consists of Object Analysis and Use Case Modelling. The former is based on the Object Modelling Technique (OMT) and the latter embraces Message Sequence Charts (MSC). Architectural Design is performed concurrently with the Test Design. The Detailed Design phase covers Data Modelling (Abstract Data Types, ADTs) and Behavioral Design. Specification and Description Language (SDL) is used as the major design method.

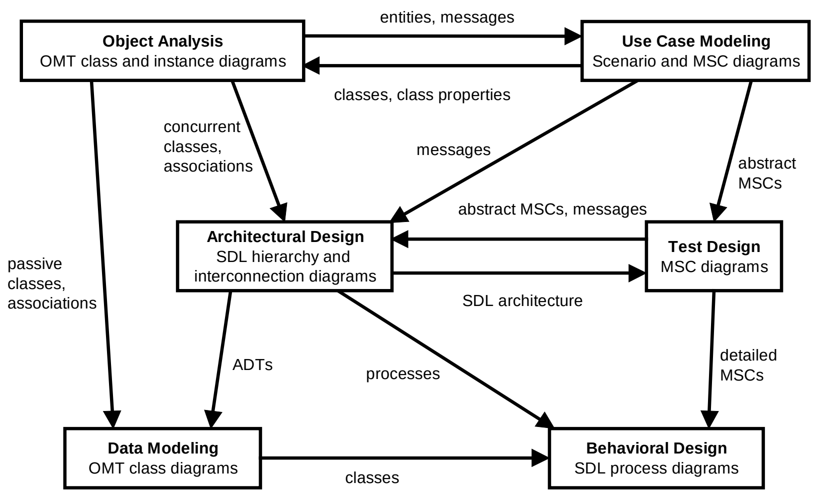


Figure 26. OORT traceability links [OORT 96].

As later discussed in sections 4.3.5 and 4.3.6, Targeting and Testing complement the software life cycle. The ObjectGEODE methodology employs an iterative development approach, resembling evolutionary and incremental life cycles, thus differing from the traditional waterfall model.

4.3.1 Requirement Analysis

Support for Object Modelling. Object modelling is weakly supported. The tool supports only class and instance diagrams, and mainly basic concepts. Data dictionaries are well supported.

Support for Use Case Modelling. Use Case modelling is weakly supported. The Message Sequence Charts make it easy to construct clear and readable scenarios for presenting interactions between a use case and actors. They can be further refined with sub-scenarios. Addressing the problem of complexity and larger number of scenarios, it is possible to build a scenario hierarchy. However, the solution has its limits and does not comply with the standard high-level MSC (HMSC).

Support for coherence between the object model and the use case model. This feature is only very weakly supported. The support focuses on dictionaries and maintaining name consistency. The semantics is not considered.

Support for relationships between Requirement Analysis and the other development activities. This feature is significantly supported. The support for other activities is available but it is also dependent on the user's discipline to follow certain guidelines, which are not mandatory tool features.

4.3.2 Architectural Design

Support for SDL. SDL is significantly supported by the tool. ObjectGEODE has a good and versatile set of SDL editors and other tools. Different views on the model and partitioning of the model are valuable features. However, not all the SDL standards are fully supported.

Support for relationships between architectural design and detailed design. This feature is significantly supported. Detailed design is essentially carried out by refining the architectural design. This includes the refinement of passive objects (ADTs and algorithms) and concurrent terminal objects (processes).

Support for relationships between architectural design and test design. This feature is significantly supported. Architectural design and test design are concurrent activities. The related SDL level consists of an interconnection diagram where an MSC can act as an observer of SDL model.

4.3.3 Detailed Design

The Detailed Design phase is significantly supported by the tool. The tool offers several design features: OMT, MSC and SDL dictionaries, extensive checking, ADT signature translation from OMT classes, importing of external objects from the ObjectGEODE environment, design-level maintenance and simulation. Here again, the user must follow some guidelines and also manually modify environment files. Writing additional code is needed at least in the implementation of SDL objects.

4.3.4 Test Design

Test Design is significantly supported by the tool. Precise Test Design can automate testing to some extent. In addition to detailed MSCs, more powerful GOAL specifications can be used as observers at the SDL level.

4.3.5 Targeting

Targeting is significantly supported by the tool. The main features include run-time libraries (RTL) for cross development platforms, partial saving and loading of build options, code generation options and instrumentation. Manual, error-prone editing of environment files and build scripts is required for integrating the make utilities and the compiler into the tool.

4.3.6 Testing

Testing is significantly supported by the tool. The instrumentation allows tracing the executable file, while the most useful feature is the design simulation (executable model). The simulation was well supported with saving and loading of scenarios, stepwise or reverse execution, playback, filtering of transitions and output, stop conditions and open models for controlling the environment behaviour. Extensive observation facilities covered states, communications, data, time, variables, transitions, events, observers and simulation coverage. The simulation is capable of detecting deadlocks, exceptions and livelocks. The mode of the simulation can be interactive or automatic (intensive or exhaustive). In addition, further useful options are available for optimisation purposes.

4.3.7 The OORT iterative process

The OORT iterative process is only weakly supported by the tool. The existing traceability links are well supported and appropriately updated, but the builder cannot always recognise changes in the model stemming from the user code or builder options. OORT iteration support is good but requires the incorporation of a revision control system. Multi-user environment and document support are fair.

4.3.8 Methodology evolution

The methodology evolution is significantly supported by the tool. The evaluated version of the ObjectGEODE supports the most important versions of SDL (SDL-88 and SDL-92), subset of OMT and partially MSC standards (Z.120). The vendor is also actively following the evolution of these methods and standards, so the effect of the UML (Unified Modeling Language) is foreseen.

4.4 ESA standards

Questions and criteria for ESA standards were derived from [ESA 1991]. The ESA standards criterion had two low-level criteria: Product standards and Process standards. The criterion is partially satisfied.

4.4.1 Product standards

Product standards are significantly supported by the tool. Different product features and software life cycle phases are supported variably. The strong points are the design phase, prototyping, milestones, definition of the capabilities, construction of the logical model, classification of the requirements, consistency, top-down design, functional definitions, data structures, control flow, stepwise refinement guidelines, code completeness, consistency and structure, integration, unit and black-box testing, system testing and maintenance at model level. Documentation support provides alternatives for flowcharts, pseudocode and test reports. The weaknesses can be found in textual user requirements, document and diagram information, non-functional capabilities, configuration management and white-box testing.

4.4.2 Process standards

Process standards are weakly supported by the tool. Better estimation would need a real-life project experience, i.e. a pilot project. Software configuration management is not embedded in the tool set, while software verification and validation are well supported. However, the support for the software project management and quality assurance depends heavily on the existing practices of the organisation. This also requires a special transition project and adoption of suitable metrics for object-oriented software development. As the emphasis lies in the practice and the application of the standards, the tool support and the processes adopted within the organisation are likely to affect each other. Hence, the capability level is given with some reservations.

4.5 Application requirements

The application requirements criterion contained miscellaneous questions that were not included in the previous criteria. They were specific to the evaluator’s organisation and application domain. The criterion is largely satisfied, but due to the diversity of the single requirements, variation in credits is relatively large.

The graphical simulation (animation), standard formats (PostScript, ASCII) and supported platforms seem to give the best value in this category. Although many of the requirements are not directly fulfilled, they are partially implemented or they can be achieved by other means. This implies the inclusion of third party tools and additional manual work. For example, the revision control system, code editor and compiler must be integrated into the tool environment. Printing large diagrams is surprisingly clumsy. The integration of user code, e.g. graphical user interface, is not the most trivial task. It will not be easy to compensate for the lack of support for real-time performance estimation and measurement.

4.6 Summary

The used three-stage approach proved to be efficient in the evaluation of the functional aspects of the ObjectGEODE methodology. The proposed evaluation framework was easy to use and focused only on the aspects considered essential and important by the evaluation organisations. The specific requirements and needs of each organisation were clearly and successfully separated from common aspects.

The ObjectGEODE methodology is a powerful tool for the expert user familiar with general object-oriented concepts and notations like OMT, SDL and MSC. Although the tool still suffers from some immaturity, the evolution of the tool and the standard notations composing the OORT method are very promising.

The used evaluation framework resulted in capability profiles concerning the support the ObjectGEODE methodology for the OORT method, ESA standards and application requirements. The interpretation of the profiles indicates clear profits achieved by the methodology. However, the final applicability within the organisation should be carefully assessed in combination with all the relevant aspects. The capability profiles presented here should be carefully assessed in combination with general tool capabilities. The methodology offers obvious advantages, but applying it efficiently is a challenging task, in which training is a necessity. Modern software practices and disciplined project management are still needed.

5. CASE Tool Evaluation

CASE tools play an essential role in the software development of embedded software. The evaluation and selection of CASE tools that best match the needs of an individual organisation is often a rather complicated and confusing process. Modern real-time CASE tools are software development environments consisting of an integrated set of different tools and informal, semi-formal and formal description techniques. The tools support the simulation and validation of formal design models and components as well as automatic generation of target code from the design models. Thorough evaluations of these tools call for extensive background knowledge and quite a lot of specific experimenting with the tools. As a general guideline, the choice of a CASE tool should be made after determining which software development methods will be adopted, because the tools are based on these methods, thus providing automated assistance to designers using these methods.

In this chapter, the application of a set of CASE tool assessment criteria to three CASE tools will be described. The resulting specific capability profiles for the individual tools are presented and analysed, focusing on the status, strengths, weaknesses and potential benefits of the tools.

The first section of this chapter is an introduction to the used evaluation approach. The second section presents the resulting specific capability profiles for the evaluated tools. The subsequent sections deal with the evaluation results of the following tool aspects: ease of use, power, robustness, functionality, ease of insertion, and quality of commercial support. The ninth section discusses some interesting issues concerning CASE tool evaluation. Finally, the tenth section summarises the outcome of the evaluation with the framework.

5.1 Introduction

A CASE tool assessment process may involve the following four steps [Firth et al. 1987]:

1. Analyse the purposes for which the tool will be used,
2. Analyse the environment in which the tool will be used,
3. Develop a list of candidate tools that may meet above needs, and
4. Apply a set of assessment criteria to each of the candidate tools and select a tool.

The tool will be used for the development of mission-critical software. Specific functional and methodological requirements for the tool are specified in Chapters 2 and 3. The requirements differ from the functional requirements used in Chapter 4. The tool has to support the simulation of design models. The tools will be used in NT or UNIX platforms.

The candidate tools were ObjectGEODE by Verilog SA, SDT by Telelogic AB, ObjecTime by ObjecTime Limited and Rhapsody by i-Logix. Rhapsody is new and proved to be not mature enough during the Space2000/sw project and its evaluation conditions were too expensive. The evaluated CASE tools were ObjectGEODE (versions 1.0.1 and 1.1), SDT (versions 3.1 and 3.11) and ObjecTime (version 4.4).

The evaluation framework used in this study is described in Appendixes A and B. The framework was adapted from [Firth et al. 1987], [Salmela 1994] and [Vierimaa et al. 1998]. Frameworks such as [Firth et al. 1987], proposed for the assessment of real-time software engineering tools, do not address tools that are actually integrated environments.

The results of these evaluations have also been presented in [Ihme et al. 1998], [Paakko & Holsti 1997a], [Paakko & Holsti 1997b], [Suihkonen 1998], [Suihkonen & Kumara 1997b], [Suihkonen & Kumara 1997c] and [Toivanen 1998a].

5.2 Capability profiles of the tools

The capability profiles of the evaluated tools are shown in Table 2. The profiles of the low and high level criteria are expressed as a percentage of the given score to the maximum score of each criterion:

$$\text{Profile}_L = (\text{Score}_L / \text{MaximumScore}_L) * 100\ \%$$

The scale numbers and maximum scores used in the individual evaluations showed slight variations. The weighting factors shown in Figure 27, expressed here as average percentages, were used for the high level criteria.

Table 2. Capability profiles of the evaluated tools as percentages of maximum scores.

	ObjectGEODE	**SDT**	**ObjecTime**
Ease of Use	**49 %**	**52 %**	**54 %**
Tailoring	33 %	33 %	33 %
Intelligence & Helpfulness	33 %	41 %	46 %
Predictability	67 %	67 %	67 %
Error Handling	48 %	52 %	67 %
Multiple Users	67 %	67 %	67 %
System Interface	56 %	56 %	44 %
Power	**44 %**	**44 %**	**45 %**
Tool Understanding	61 %	61 %	61 %
Tool Leverage	42 %	42 %	42 %
Tool State	24 %	24 %	24 %
Performance	48 %	48 %	52 %
Robustness	**54 %**	**54 %**	**55 %**
Consistency	71 %	71 %	57 %
Evolution	61 %	61 %	70 %
Fault Tolerance	50 %	50 %	50 %
Self Instrumentedness	33 %	33 %	47 %
Functionality	**66 %**	**66 %**	**54 %**
Methodological Support	63 %	63 %	47 %
Correctness	73 %	73 %	73 %
Ease of Insertion	**61 %**	**63 %**	**60 %**
Learnability	55 %	59 %	61 %
SW Engineering Environment	67 %	67 %	60 %
Quality of Support	**64 %**	**67 %**	**62 %**
Tool History	58 %	58 %	42 %
Vendor History	68 %	67 %	67 %
Purchase, Licensing, Rental	63 %	63 %	52 %
Maintenance Agreement	69 %	69 %	69 %
User's Groups & Feedback	67 %	75 %	58 %
Installation	78 %	67 %	78 %
Training	54 %	67 %	58 %
Documentation	54 %	71 %	70 %

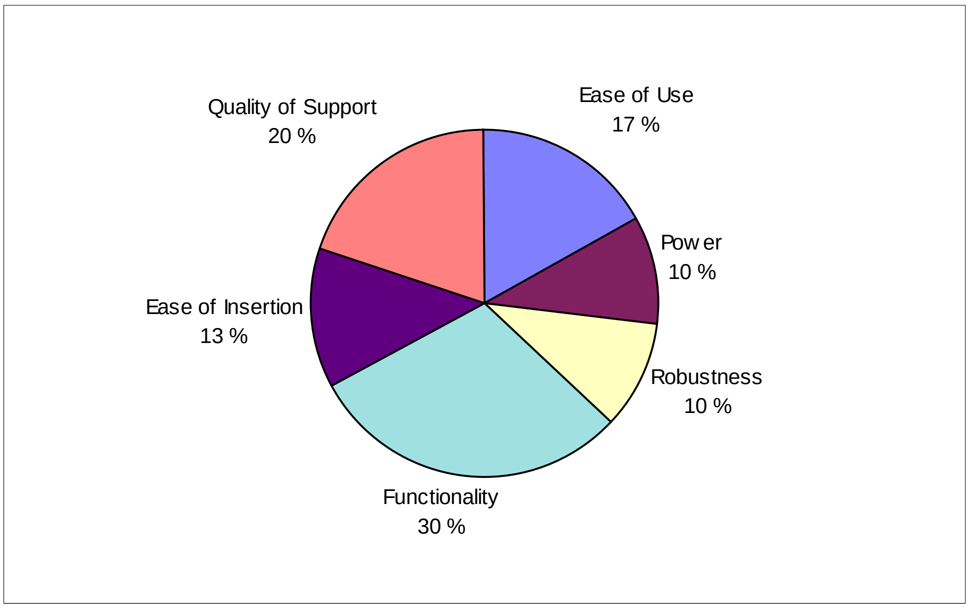


Figure 27. The average weighting factors of the high level criteria.

The percentages of the capability profiles were converted to the following four textual credits of the criteria using the following rules:

Textual Credit	Percentage
Not satisfied	0 – 39.9
Partially satisfied	40 - 59.9
Largely satisfied	60 – 79.9
Fully satisfied	80 - 100

The rules differ slightly from the rules used in [Ihme et al. 1998]. The textual credits are used in the next sections for interpreting the numeric capability profiles.

5.3 Ease of use

The criterion ”ease of use” is partially satisfied by the tools as shown in Table 2. The evaluated tools support the properties included in this criterion as follows:

5.3.1 Tailoring

The answers to the tailoring questions indicate how well the tools can be tailored to the varying needs of a wide range of organisations and users. It may be essential for CASE tools and particularly code generators for on-board software that they support adapting to various specific purposes. This feature is only very weakly supported by the tools.

5.3.2 Intelligence and Helpfulness

The tool should help the user, anticipate user interaction and provide simple and efficient means of executing the functions required by the user. This feature is only weakly or very weakly supported by the tools. The tools cannot be used without comprehensive training. Only inadequate help is provided by ObjectGEODE and SDT for the object-oriented features of SDL and code generation. For navigating between model structures, the poorest support is provided by ObjectGEODE and the best by ObjecTime. ObjecTime allows editing the design models that are loaded for simulation. The most context-sensitive on-line help is provided by ObjecTime.

5.3.3 Predictability

Unpredicted responses are likely to result in unwanted output and unhappy users. This feature is significantly supported by the tools.

5.3.4 Error Handling

The tool should tolerate user errors and it should be able to check for and correct these errors whenever possible. This feature is significantly supported by ObjecTime. SDT and ObjectGEODE support this feature weakly. The error correction of the object-oriented features of SDL models is only very weakly supported by both ObjectGEODE and SDT.

5.3.5 Multiple Users

The multiple users feature denotes that a tool can be used by many users in a project. This feature is significantly supported by the tools. The tools were not evaluated for extensive long-lived projects, in which some problems had been encountered elsewhere.

5.3.6 System Interface

The system interface feature denotes the capability of interfacing with other tools, important data formats or devices. The tools support this feature weakly. The portability of SDL-88 text files between ObjectGEODE and SDT works well in practice. Some problems were encountered here, but these could be resolved. Unresolved portability problems of SDL-92 text files were, however, encountered between ObjectGEODE and SDT. The object models and MSCs are not portable between the tools. In addition, there are notable deficiencies in the interfaces with project documentation tools.

5.4 Power

The Power criterion is partially satisfied. The tools support the properties of this category as follows:

5.4.1 Tool Understanding

The characteristic of tool understanding stands for the capability of a tool to manipulate objects that have an inner structure. It is important for the tool to understand the object content and structure and to be able to handle different aspects of that structure. The tools provide a significant support for this feature. ObjecTime is capable of analysing and manipulating collections of objects in hierarchical ROOM models. ObjectGEODE and SDT are able to handle and analyse different aspects of the SDL structures. The capability of analysing MSCs and OMT models is, however, more limited.

5.4.2 Tool Leverage

The leverage of the tool denotes the extent to which small actions or commands by the user can produce extensive effects. This feature may also imply that commands can be “overloaded” so that one command name may have different implementations for different objects, while commands may also be inherited and composed. The tools support this feature only weakly.

5.4.3 Tool State

If the tool remembers how it has been used in a current session or in previous sessions, it can provide the user with simple ways of enhancing the power of the tool. This feature is only very weakly supported by the tools.

5.4.4 Performance

The performance of a tool affects the ease of use, thus it may determine the success of the tool within an organisation. The tools provide a weak support for this feature.

5.5 Robustness

The Robustness criterion is partially satisfied. The tools support the properties included in this criterion as follows:

5.5.1 Consistency of the operation of the tool

This feature is significantly supported by ObjectGEODE and SDT and weakly by ObjecTime. ObjectGEODE and SDT support the standard design language SDL. The ROOM language of ObjecTime has not been standardised.

5.5.2 Evolution

A tool evolves to accommodate itself to changing requirements, changes in the environment, corrections of detected flaws, and performance enhancements. The tools support this feature significantly.

5.5.3 Fault Tolerance

The tools equally provide a weak support for the fault tolerance specifically related to the tools.

5.5.4 Self Instrumentedness

A tool should be self-instrumented to be able to assist in determining the cause of a problem or a bug once symptoms have been detected. This feature is weakly or very weakly supported by the tools.

5.6 Functionality

Functionality is the most important criterion for the CASE tool selection. The criterion is largely satisfied by ObjectGEODE and SDT and partially by ObjecTime. The tools support properties included in this criterion as follows:

5.6.1 Methodological Support

Both ObjectGEODE and SDT show a significant capability of automating and supporting methodologies associated with the organisations. The ObjecTime version 4.4 supports this feature only weakly, because the size of the C++ code generated by ObjecTime does not fulfil the mandatory requirements for on-board software. There is also a code generator producing C code for small target systems available for ObjecTime.

The most important characteristics of the generated code are code size, execution speed and the ease of interfacing to other parts of the system. Typically, the size of the generated code is always larger than that achieved with hand coding. The sizes of the corresponding C code generated by ObjectGEODE and SDT from the SDL model of the SIXA Measurement Controller in comparison with the size of the hand-coded code is shown in Table 3. The same C compiler was used for the generated C codes. The size of the hand-coded code is calculated from the actual implementation of the SIXA instrument software.

Table 3. The size of the generated code of the SIXA Measurement Controller in comparison with the size of the hand-coded code.

Code Generator	Code Size in comparison with hand-coding
SDT/Cadvanced	fourfold
SDT/Cmicro	twofold
ObjectGEODE	threefold

The speed of the generated code can easily be as high as that of a corresponding hand-coded code, provided that appropriate coding rules are used. The interfacing to other parts of the system (e.g. operating system timer and message passing primitives) is typically not uncomplicated. Although the characteristics of the C code generated by ObjectGEODE and SDT are not very satisfying, the code can be used in target systems to some extent.

The tools do not support the domain analysis phase. No external event lists are supported, either. Furthermore, the textual requirements are necessary, but they are not adequately supported by the tools. There are shortcomings in the integration of textual descriptions and graphical models with the specification and design documents.

The SDL language does not contain any real-time task definition properties. SDT supports the assignment of SDL processes and blocks to tasks. ObjecTime supports synchronous and asynchronous communication and message priorities. Optional actors can be assigned to different independent logical threads of execution. Logical threads may be assigned to different actual physical thread configurations for generating target implementations.

The object-oriented features of SDL are specific to SDL. ObjectGEODE and SDT do not support the use of object-oriented languages in the implementation of SDL models. It is difficult to apply sound object-oriented design principles and patterns in SDL design using ObjectGEODE or SDT. Some object-oriented features of ROOM, such as behavioural inheritance, are specific to ROOM supported by ObjecTime. The tools do not support the description of design patterns.

SDL models can be tested using the simulation and validation capabilities of ObjectGEODE and SDT. ROOM models can be validated using the ObjecTime simulator. ObjecTime includes only the simulator and no validator tool. In the ObjectGEODE and SDT environments, the use of coverage tools will further enhance the testing phase.

The disadvantages of SDL and ROOM include inadequate support for complex data type, algorithm and hardware interface descriptions. The number of pre-defined C++ data types in ObjecTime is too small. The weak support of SDL for data descriptions may cause problems in the automatic system validation process. The data structures and algorithms may be defined outside SDL with the C language. However, if these C functions are called from within the SDL model the validator tool will have no control over the activities and data structures implemented in C code. Due to this, parts of the system state space will be hidden from the validator, and thus the automatic validation process will not work properly.

The tools do not support the timing constraints and performance analysis of the system. Nor do they support any systematic approach to reuse-driven software engineering.

5.6.2 Correctness

A tool must operate correctly and produce correct outputs. This feature is significantly supported by the tools. There are, however, portability problems with SDL-92 text files produced by ObjectGEODE and SDT. The code generators of the tools should be qualified for mission-critical software.

5.7 Ease of Insertion

The Ease of Insertion criterion is largely satisfied by the tools. Learning how to use the tools was not very easy for the users in the organisations performing the evaluations. The tools fit well the software engineering environments of the evaluation organisations.

5.8 Quality of Support

The Quality of Support criterion is largely satisfied. The tools provide a significant support for the following properties of this criterion: Vendor History, Maintenance Agreement and Installation. Tool History is only weakly supported by the tools. There was not enough material on the use of ObjectGEODE and SDT in the domain of mission-critical controller software and no material at all on the use of ObjecTime in this domain. Neither is there any complete list available of the ObjecTime users that have purchased the tool.

Run-time licenses have to be paid for applications produced by ObjecTime. Deficiencies were encountered in the user support of ObjectGEODE and SDT concerning code generation problems. In effect, the user support of ObjectGEODE and SDT failed to respond to the inquiries on the portability problems of SDL-88 and SDL-92 text files.

There were quality problems in the training support of ObjectGEODE. The tutorials of the tools proved to be useful, particularly that of SDT. ObjecTime, again, provides a set of useful modelling examples.

5.9 Discussion

The evaluated CASE tools are software development environments consisting of an integrated set of various tools and informal, semi-formal and formal description techniques. The ObjectGEODE toolset includes editors for the OMT, SDL and MSC notations. Project Organizer, the simulator, validator and code generators can be seen as separate tools. The SDT toolset consists of tools corresponding to the above. ObjecTime has no OMT editor, but is otherwise similar. Each tool in the software development environments could have been evaluated separately using the proposed framework, thus resulting in more accurate and comparable results. However, these separate evaluations would have caused a lot of extra work. In addition, the integration of the tools in each toolset should have been evaluated. There are disturbing internal inconsistencies in the integration of the OMT and SDL tools in the ObjectGEODE and SDT toolsets.

Industry is presently showing great interest in standard description techniques. ObjectGEODE and SDT support the standard design language SDL. However, they support only those features of OMT that can easily be mapped into SDL designs. This will probably constrain the support of the tools for the de facto standard UML [UML 1997] in the future. UML is the next version of the OMT notation.

Most of the essential specification and analysis models could be modelled with ObjectGEODE, SDT and ObjecTime methodologies. The object-oriented analysis models of ObjectGEODE and SDT are useful, but they cannot be regarded as the best foundation for the object-oriented design structure. To be competent, object-oriented design must be based upon the decoupling facilities of polymorphism and encapsulation. This will make object-oriented software more flexible, maintainable and reusable. It has proven difficult to apply good object-oriented design principles and patterns in SDL design.

In addition to the simulation and validation support of design models, the possibility of generating the final application code from the design models is a very tempting feature. However, various difficulties were encountered during the experimentation with the code generators of the tools. The code generators of the tools should be qualified for mission-critical software.

5.10 Summary

An important part of the presented evaluation framework - the one hundred and fifty evaluation questions - could not be described in this paper. The classification of the questions into six high-level aspects and twenty-six low-level aspects of CASE tools was presented. The evaluation criteria of the tools were associated with the aspects.

The evaluated ObjectGEODE, SDT and ObjecTime tools are sound, mature and powerful real-time software development environments. Most of the essential aspects are supported by the tools. There are no great differences in the capability profiles of the tools. Particularly ObjectGEODE and SDT were found to be very similar. There are, however, differences in details recorded in the answers to the evaluation questions.

These differences often compensate for each other and thus cannot always be seen in the presented capability profiles.

The shortcomings of the tools include poor efficiency and quality of the generated code, lack of support for complex data type and algorithm descriptions, as well as weak performance evaluation of the system, concurrency description and hardware interface description. Due to these shortcomings, the tools can be used in system design and implementation only to a limited degree.

It is essential for the tools to be capable of automating the design of mission-critical software to the greatest possible extent. The benefits of the simulation and validation of design models are so convincing that the use of the tools is worth considering even if the final coding or module testing should be done manually due to the poor quality and efficiency of the generated code.

The experience gained in using of the evaluation framework has proven the value of the framework as a checklist, while this does not require too much extra work, either. Providing complete answers to the evaluation questions calls for extensive background knowledge and a quite lot of specific experimenting and testing with the tool.

The presented capability profiles are specific to the needs and priorities of the organisations involved in the evaluations. The framework has to be adapted to each organisation separately. The CASE tool selection process will be facilitated by the use of the framework, yet it does not provide and must not be used as an absolute recipe for choosing CASE tools.

6. Product data management

This chapter introduces the concept of product data management (PDM) and includes a study of PDM in the context of mission-critical applications using space applications as an example. The chapter has been edited from [Toivanen 1998b].

In Section 6.1, the concept and components of PDM are defined and the benefits of PDM are shortly introduced. Section 6.2 discusses the connections of product, production process and PDM components as well as defines the most important components of PDM in the context of mission-critical applications. Section 6.3 presents some suggestions of applicable PDM practices based on the priorities set in Section 6.2.

6.1 Introduction to product data management

The term PDM is a very broad concept, for which there is no universally applicable definition. Typically, PDM is defined as a software-based technology used for managing any product-related information and processes.

PDM does not create any new product information. The sole purpose of PDM is to help the user to control and manipulate the vast amounts of information and procedures connected with product marketing, design, manufacturing and maintenance. PDM is a all-inclusive support system for any product-related activities.

6.1.1 PDM user functions

In order to examine PDM more closely, the concept has to be divided into smaller components. The partitioning used in this document is presented in Figure 28 and it is based on [CIMdata 1996]. The division of components under the headings “Process management” and “Data management” is based on [PDMIC 1997].

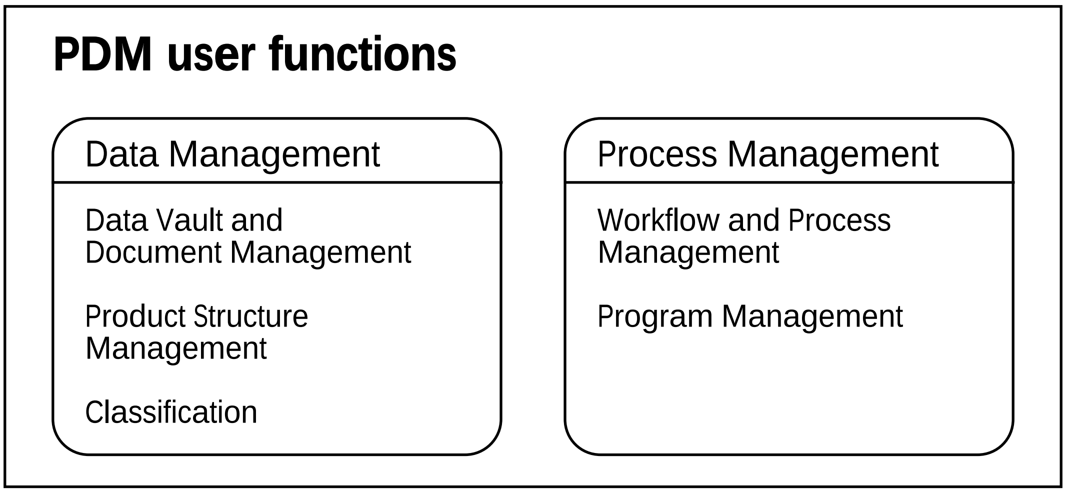

PDM user functions
Data Management
Data Vault and Document Management
Product Structure Management
Classification
Process Management
Workflow and Process Management
Program Management

Figure 28. PDM user functions.

The PDM user functions are introduced shortly below. For more details, refer to [CIMdata 1996] and [PDMIC 1997].

Data vault and document management is responsible for secure, controlled handling and storage of all product data. Data is stored in the data vault either directly (documents in electronic form) or by reference (non-electronic documents, e.g. papers and microfilms).

Product structure management facilitates the creation and management of product configurations. Different types of data (drawings, test data, component information etc.) can be linked to the product structure so that the relations between associated data elements can be seen.

Classification of parts allows similar or standard parts, processes and other design information to be grouped according to common attributes. The classification also provides efficient mechanism for finding parts according to user needs. This promotes the re-use of the work already done.

Workflow and process management is used for defining the repetitive processes of the enterprise. After the processes have been defined, the workflow management ensures that work is being done according to the defined rules.

Program management provides work breakdown structures and enables resource scheduling and project tracking. The provided capabilities are similar to project management systems.

6.1.2 PDM utility functions

PDM utility functions provide support for facilitating the use of the PDM system and the user functions introduced above. The utility functions are listed shortly below. The list is based on [CIMdata 1996].

Communication and notification support means that all the personnel is informed concerning the current state of the project. Communication is typically carried out via email. Automatic notification can streamline the workflows e.g. by sending email to the document approver after a document has been checked in to the data vault in the “proposal” state.

Data transport functions isolate the user from the computer and network environment. User does not need to know where the data is stored in the network or how the file and directory commands of certain operating system are used.

Data translation functions convert data between pairs of applications. This allows enforcement of standard data forms. Automatic data translations relieve the user of knowing which translator to apply in each case.

Image services provide the chance to treat raster, vector and video images the same way as any other data.

System administration services are used to set up the operational parameters of the PDM system and to monitor its performance.

6.1.3 Benefits of PDM

Process management utilities strengthen the use of standard processes in product data handling. Typically, also the tools (e.g. word processors, CAD systems etc.) to be used in processes are standardised. The standardisation of processes make them easier to repeat and improve. The standardisation of tools makes the data more accessible, because the need for data format conversions is reduced.

The data management utilities make all product data immediately accessible to those who need it, thus enabling concurrent engineering. All changes are immediately spread throughout the organisation. Also the history of the data is recorded, making the return to previous versions easy e.g. in case of failure of a new design approach. The structuring of product information - both by classification and product structures - make the finding of required data element easier. This facilitates the re-use of existing information both on component and architecture level and reduces the danger of wasting time on inventing the same things over and over again (“re-inventing the wheel”). The capacity of defining different views to data is very likely to consolidate the understanding of the whole product and to facilitate controlling the product data integrity.

6.2 PDM and mission-critical applications

PDM is a very large concept. Different PDM components are important to different enterprises. In this section, the relations between the product characteristics, production process and PDM components are studied. The treatment of the subject is not by any means complete. The idea is to bring out several views to the nature of a product and the production process and to study the corresponding requirements for a PDM system.

Then a typical lifespan of a mission-critical space application development project is introduced. Finally, the focal areas of PDM from the point of view of mission-critical applications are defined.

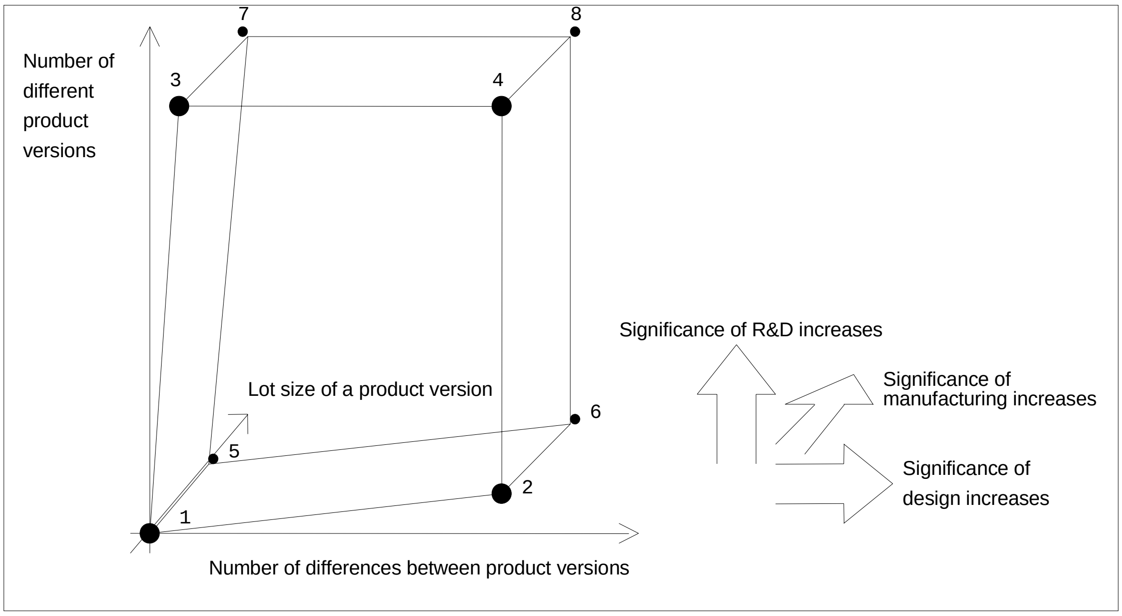
6.2.1 Characteristics of the product

One possibility to view the characteristics of the product is to examine the number of the different versions of a product, the number of differences between product versions and the product version lot size, as presented in Figure 29.

The *number of differences between product versions* describes how much the different versions of a product differ from each other. The *number of different product versions* describes the total number of different versions. Depending on the product, different versions may originate from internal continuous development (e.g. new features are added to generic GSE (Ground Support Equipment) system and new releases of GSE are delivered) or from customisation according to customer needs (e.g. tailoring a generic GSE system to fit the exact requirements of the customer). *The lot size of a product version* describes how many pieces of each product version are produced. The total number of produced products is given by the formula

$$\sum_{1}^{n} \textit{lot size of product version}_{\mathrm{n}}$$

where n is the number of different product versions.



7
8
Number of different product versions
3
4
Significance of R&D increases
Significance of manufacturing increases
Lot size of a product version
6
5
Significance of design increases
2
1
Number of differences between product versions

Figure 29. Characterization of a product.

The term *product version* is used in this document in the broadest sense of the word. For example, the GSE systems of different satellite instruments are considered versions of a single product, if some re-use exists between different GSE systems.

Figure 29, shows a chart with some example products, as follows:

- Product 1 is a "one-of-a-kind" product. Only one version of the product is made. A typical example is a scientific instrument that is used in a satellite. Only one device is manufactured, and due to the specialised nature of the instrument, none of the software or hardware can be re-used.

- Product 5 is a "one-of-a-kind" product that is mass-produced after the designing. A typical example is a memory expansion card to a discontinued computer system. The card is designed to give some additional service life years to a system that would otherwise become obsolete.

- Product 2 is a largely customised product, but the number of different product versions is quite small. An example will be provided by the GSE system of a satellite instrument. Only a few GSE systems are delivered in a year. The hardware and software of the GSE have to be customised to fit the target environment.

- Product 6 is a customised product that is mass-produced. A typical example is a cellular telephone network base station. The base station is customised for the operator and then hundreds of devices are manufactured and installed.

- Product 3 is a customised product, but the different versions are quite similar. A typical example is a car. When a car is ordered the customer defines the colour, motor size, accessories etc. of the car based on a fixed set of choices. The car is manufactured at the factory according to the customer instructions and then delivered to the customer. The customisation is carried out by having a predefined generic product structure, which is then specified according to customer needs. Technically, the customisation is founded on a modularisation of the product, allowing the customer to select a module (e.g. the motor of the car) from a set of choices.

- Product 7 could be represented by a car type ordered by a rent-a-car company. After customisation hundreds of similar cars are produced, so as to minimise maintenance costs and to maximise purchasing discounts.

- Product 4 is a mass-customised product. Both the number of different versions and the differences between the versions are great. A typical example is provided by an elevator. An elevator factory may produce thousands of elevators in a year, while all the elevators are different. The features of an elevator depend on the properties of the target building and the wishes of the customer, and both hardware and control software have to be specially built for each delivery.

- Product 8 is a mass-customised product with large lot sizes. A typical example is represented by a valve used in the pipe system of a power plant. The valve has been specially designed according to the needs of the power plant, while hundreds of similar valves are used in the power plant.

If the product is not of the one-of-a-kind type, some re-use of components will occur between product versions (or between different products). Here the term *component* is used in a broad sense denoting that e.g. software units, hardware units and architectures as well as production processes are considered components of a product. When a component of a product is examined it can be grouped according to the framework presented in Figure 30.

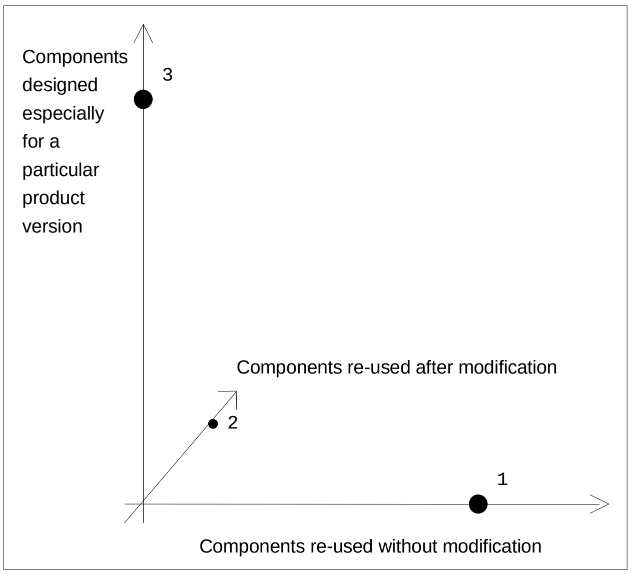

Components designed especially for a particular product version
3
Components re-used after modification
2
1
Components re-used without modification

Figure 30. Characterization of a component.

In Figure 30, component 1 is re-used as it is in the new product version. Operating system kernels and commercial electronics boards represent typical components of this type. Component 2 is re-used after modification. Typical components of this type are software object classes or circuit diagram segments. Component 3 is specially designed for a particular product version. Typical components of this type are special software modules and electronics that are used e.g. to process data received from a special sensor type in an on-board space application.

In this document the word *customisation* stands for the production of a new product by modifying an existing product or product structure description. The modification is performed by altering the existing components (y-axis of Figure 30) and by adding new components to the existing configuration (z-axis of Figure 30).

The larger the number of product versions, the more important the role of re-use becomes. Even the components that seem to be unique for a particular product version should be designed according to rules that enable re-using the component later.

Another important product characteristics is *product complexity*. Although it is hard to measure the complexity, metrics such as parts count, relevant technology areas (e.g. electronics, software, mechanics) and the amount of documentation can be used.

Product lifespan should also be considered. In case of durable commodities, the maintenance phase of the product may be more important than the design and production phases. For example, the lifespan of an industrial crane may well be 10 to 30 years. The profit earned from the selling of the crane is minor compared with the profit of service and maintenance operation. In the course of time, parts of the crane may be modernised i.e. the product changes also after the delivery.

6.2.2 Characteristics of the production process

Depending on the nature of the product, also the production processes differ. In case of a "one of a kind" product, the process consists of requirements analysis, system design, manufacturing and maintenance, as described on the left hand side of Figure 31.

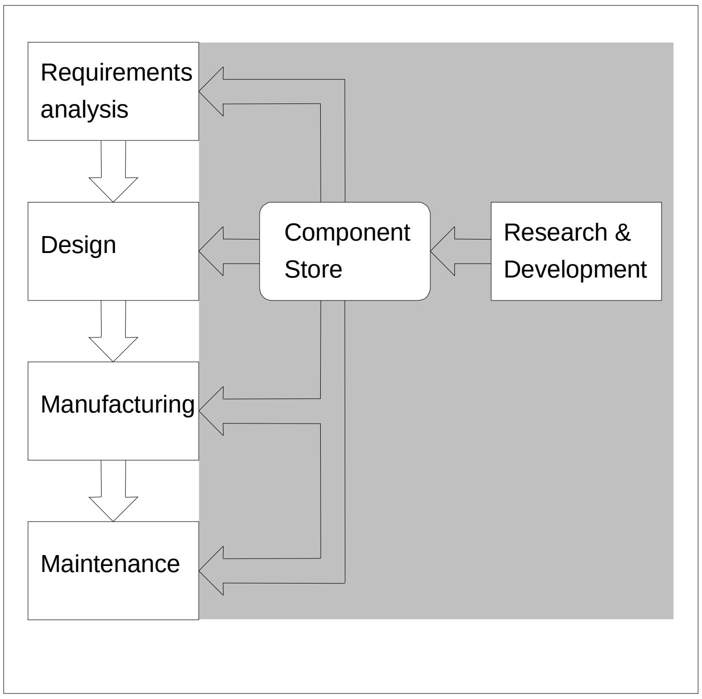


Figure 31. Production processes.

If several versions of the product are produced, the shaded area on the right hand side of Figure 31 becomes relevant. The re-use of earlier effort is achieved by having a component store that is updated by research and development function. The *component store* is to be understood as a general term here, denoting the storing of all product-related information, including design (product structures, electronics, mechanics and software component and subassembly information etc.), manufacturing (manufacturing procedures and tools, CNC programs etc.) and maintenance information (problem reports, changes made after delivery etc.).

The design, research and development functions of Figure 31 are often carried out by the same people and it is thus hard to distinguish the functions from each other. Some of the components generated in the design phase can be of general use, and thus they are added to component store. In this document, the terms design, research and development are defined so that *design* is responsible for designing of a single product version whereas *research and development* (R&D) maintains and develops reusable product data.

The significance of the component store depends on the nature of the product. In case of an one-of-a-kind product the use of component store is non-existent or only some general components (e.g. power sources or general software libraries) can be re-used. In the extreme case of the mass-customised product, the system design phase plans the product entirely based on ready-made components. With proper computer aided tools, the design process can even be partially or completely automated. Other cases lie somewhere between these extremes.

Products that have a small number of product versions are typically design-orientated i.e. the design function is predominant. If the number of product versions is large, both design and R&D are important. The design process must operate efficiently and rapidly, which requires a well-designed component store and active R&D activity.

Another view on the production process is provided by the *size and location of the project group*. New product versions are typically developed in projects. The size of the project group may vary depending on the particular phase of the project. In international companies, the project group may be geographically scattered world-wide. Also the nature of the project - whether only a new, slightly modified version of the product or a completely new product family is being developed - has a strong effect on the size of the product group. The complexity of the product is often a predominant factor, because the designing of a multidisciplinary product will require specialists from all the relevant technological areas.

From the organisation point of view also *the number of concurrently active development projects* is important. In case of mass-customised products (e.g. elevators), there may be dozens or even hundreds of development projects running simultaneously. On the other hand, a company specialised in on-board space instruments may have only be involved in one or a few projects at a given moment.

6.2.3 Product, process and PDM components

The purpose of PDM is to help the user to control and manipulate the information and procedures connected with product design, manufacturing and maintenance. Depending on the nature of the product and production process, different PDM components will be emphasised. Typically, all PDM components are useful, but due to limited resources companies have to focus on key components when the PDM system is set up.

The mapping of the needs of the enterprise to PDM components must be done for each case separately based on the specific needs of the enterprise. A very coarse mapping of relations between PDM components and the characteristics of the product is presented in Table 4, which serves as an example of an analysis that has to be carried out in a more detailed way during the evaluation phase of the PDM concept.

Table 4. Characteristics of the product and components of the PDM.

	Data Vault and Document Management	Product Structure Management	Classification	Workflow and Process Management	Program Management
Lots of different product versions	√	√	√	√	
Lots of differences between versions			√	√	
Large lot sizes			√	√	√
Complex product	√	√		√	
Large project group	√			√	√
Lots of concurrent projects				√	√

If there is a great number of product versions, almost all PDM components will be important. Data vault and document management are needed to store and organise the extensive documentation. Product structure management is a key factor during the design phase of new product versions. Classification helps in standardisation and finding parts used in the products. Workflow management defines and helps to control the design and manufacturing processes.

If the product versions differ a lot from each other, a large-scale re-use of the product data is not possible, thus reducing the importance of the data management functions. Classification may be important in enabling an efficient utilisation of occasional possibilities of re-use. The process management - i.e. the definition of processes used in designing a new product - become important.

If the produced lot sizes are large, the manufacturing point of view has to be emphasised in the design phase. The logistic function (e.g. availability of required components in store) and the scheduling of the production are important. Classification, process and program management are the focal areas in this case.

If the product is complex, data management functions are essential for organising and finding product information. Typically, a complex product is also complicated in design and manufacture, thus making the process management view important, too.

In case of a large project group, the document management system is required so as to offer the whole project group an up-to-date view of the product data. Process and program management ensure that the project group works efficiently and the progress of the project can be controlled.

If several concurrent projects are to be managed, the process management and program management components of PDM are of great importance.

6.2.4 Typical lifespan of a mission-critical application development project

A simplified process diagram of a typical mission-critical space application development project has been outlined in Figure 32.

Figure 32 offers a partial view of the technical activities involved in a typical project. The process is divided into system level activities and technology-specific activities. Depending on the nature of the device some of the activities may not apply to project (e.g. pure software or hardware development projects). Project management, change request processing, component purchasing, prototype development and other important activities have been left out of the figure.

Typically, both the process model and the contents of project activities are strictly controlled by the standards of the customer organisation (e.g. ESA or NASA). The phases of the project, deliverable items and their contents are listed in these standards. During the contracting phase, the exact contents of the project - the adapting of the standard to the particular project - is agreed on.

The project is typically divided into phases. At the end of each phase a review is held, in which the customer reviews and accepts the results of the current phase. After successful review, the project moves on to the next phase. Accepted deliverables - e.g. documents and software - form a *baseline*. After the testing of a system, the system is transferred to users as a *release* of the system.

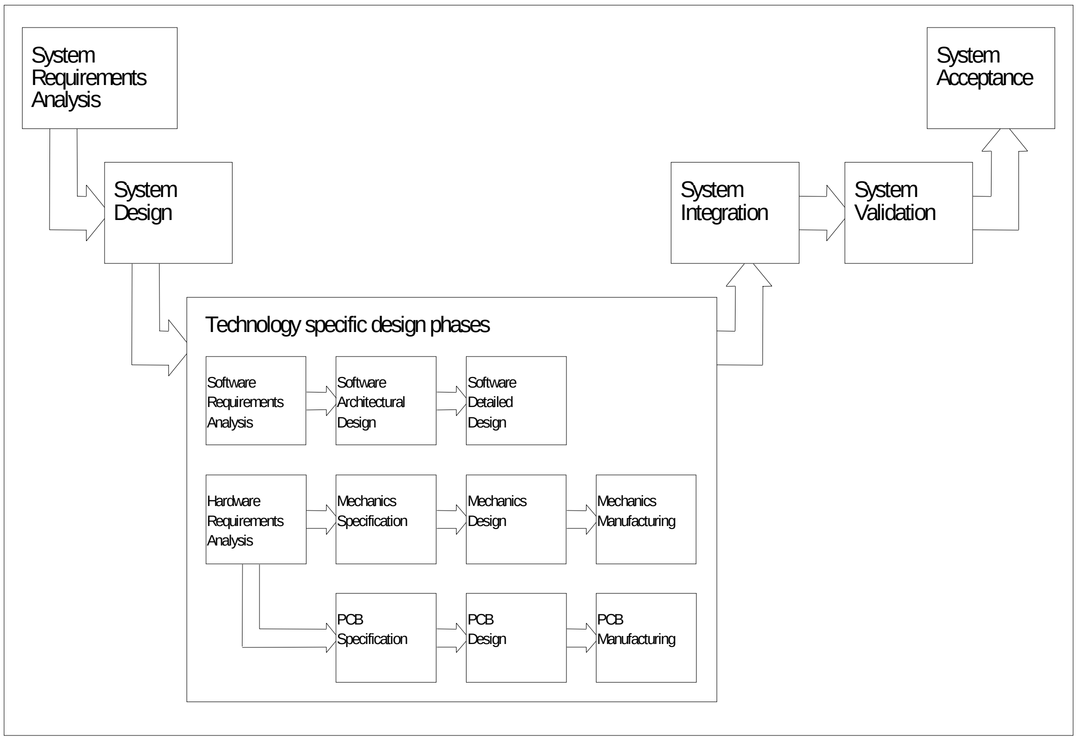


Figure 32. A simplified process diagram of a typical mission-critical space application development project.

A development project produces large amounts of documentation. The documentation plan of a particular project is prepared at the beginning of the project. In the case of space devices, the reliability and traceability requirements further emphasise the significance of documentation. Documents can be divided into *technical documentation* (e.g. system requirements document, system design specification, software requirements document, software architectural design document etc.), *plans* (project management plan, configuration management plan, quality assurance plan etc.) and *reports and forms* (change request, problem report, review item discrepancy etc.). The documentation is highly heterogeneous, including documents by mechanical, electrical and PCB (Printed Circuit Board) CAD systems as well as CASE, word processor and spreadsheet tools.

6.2.5 Focus areas of PDM in mission-critical application development

When evaluating a typical mission-critical space application according to the framework presented in Figure 29, the product will be either of type 1, close to type 2 or somewhere between 1 and 3. The application may represent the one-of-a-kind type (e.g. on-board device) or the type comprising only a small number of (remotely) related devices (e.g. GSE devices). The devices are typically rather complex and the project group size varies from small to medium. The number of concurrent development projects in one organisation is typically small.

Based on the discussion above, the competitive capacity of an enterprise delivering mission-critical applications is based on the following factors:

- **Process mastery:** The applications that are developed may differ, while the development process is quite stable. The mastery of the process - including both technical and management-related aspects - plays the key role.

- **Application domain knowledge:** The individual applications may differ, but the enterprise may specialise in a certain application domain. At its best, the application domain knowledge cumulates into re-usable application components (e.g. software and hardware modules).

In respect of the properties of the devices and the development process, the PDM components can be grouped into three categories. These categories are (in order of importance):

1. *Data vault and document management*: In mission-critical applications, the role of complete and up-to-date documentation is crucial. The document management system takes care of the version and configuration control tasks. This facilitates the maintaining of information integrity and baseline management. The common data vault also enables concurrent engineering by providing an up-to-date view on the whole project documentation to all project group members. This is extremely important, if the device under development contains both software and hardware components. The execution of workflows can also be made more effective e.g. by automating the document approval processes.

   Workflow and process management: The implementation of the PDM system is not possible, if the processes and workflows of the organisation have not been defined. In addition to process definition, workflow management can facilitate the execution and monitoring of workflow execution. However, due to the typically small number of concurrent projects (and workflows), the automatic execution and monitoring functions are not very important.

2. *Classification and Product structure management*: Depending on the type of device, either classification or product structure management is emphasised (or both). Classification is more important in the case of one-of-a-kind products, where only some general components can be re-used. In the case of more stable products (e.g. GSE-systems), on the other hand, the product structure management may be more important. Product structure management can also be used for providing different views on the product data (e.g. covering the data needed for a critical design review or the contents of acceptance data packet). Note that the classification need not concern only hardware components; software components can also be classified and stored by means of the PDM system.

3. *Program management*: The mission-critical applications are developed in projects in which project management is a mandatory practice. However, the number of concurrent projects in an organisation is typically small and the projects are of medium size. This makes it possible to manage projects with ordinary project management tools, supported by other PDM components (e.g. document and workflow management). Therefore, the program management features of the PDM system are not of high priority here.

The categories and their priorities are based on the general characteristics of mission-critical application development projects presented above. Priorities may be set differently in individual cases.

6.3 PDM tools for mission-critical applications

In this section the PDM practices and tools are studied. To begin with, the concepts of product data management systems and product data model systems are introduced. Then, a general survey of the presently available PDM tools is presented.

6.3.1 Product data management and product data models

There are two approaches to product data management: Product data management systems and product model systems [Halttunen & Hokkanen 1995, p. 33]. Product data management (PDM) systems are intended for the management of heterogeneous information environments. Product data model systems, on the other hand, operate in homogenous environments based on a single product data model. The differences between PDM and product data model based systems are illustrated in Figure 33.

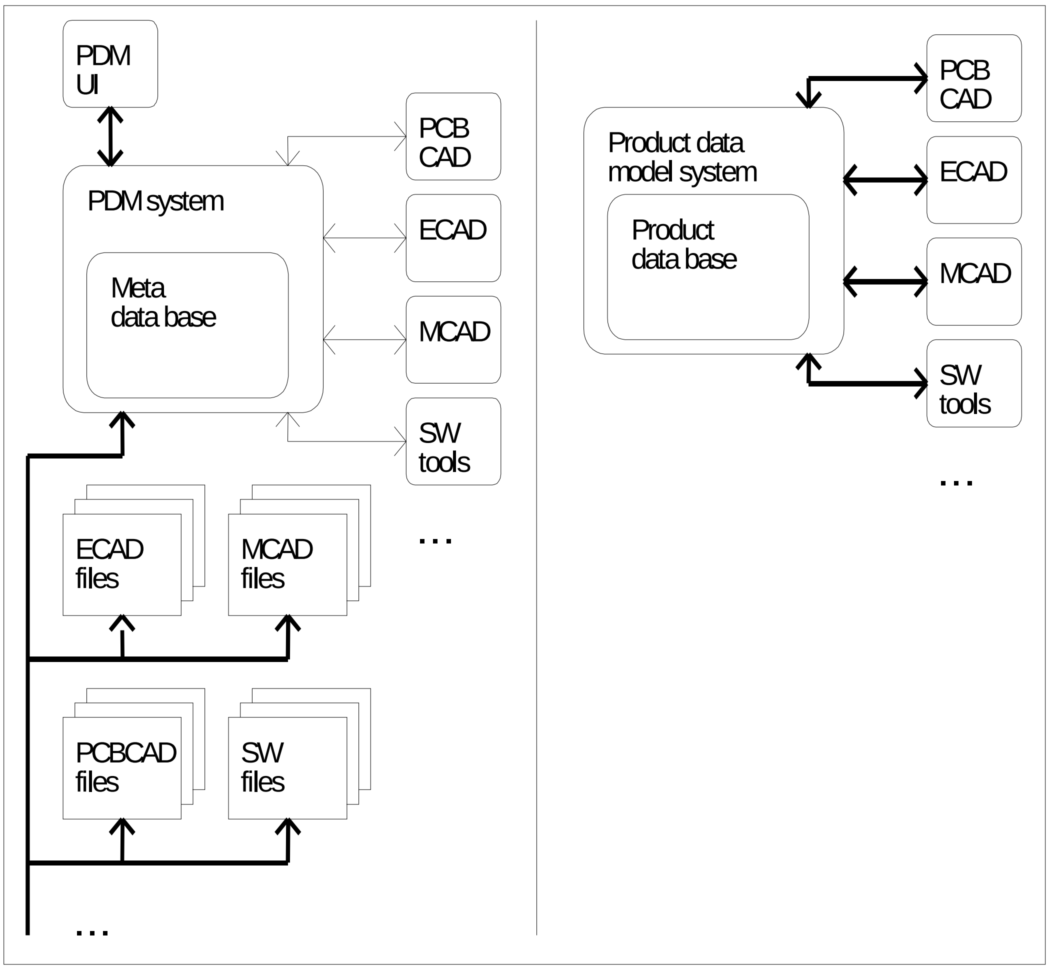


Figure 33. PDM and the product data model system.

The PDM system controls data produced with separate CAD, compiler, word processing etc. tools. Typically the PDM system only contains meta data (e.g. location information, version information) about the information elements. The actual data elements are stored e.g. in files or in paper archives. The PDM system manages the information on the statuses of and relations between these data elements. However, the PDM system does not affect the contents of data files. If two files contain overlapping information (e.g. circuit diagram and corresponding parts list) the PDM system will not be capable of ensuring data integrity between these files. The PDM system is operated through a specific user interface (UI) or via direct application software links.

The product data model system controls one product model data base (or a few) containing all the product data information. Both the product data and the corresponding meta data are stored in a common database. The product data model can be modified by application programs via application interfaces. Different views of the product data model (e.g. a circuit diagram view for an electronics engineer and a parts list view for a purchasing engineer) can be defined for different purposes. All the views utilise the common product data model.

The problems with product model systems mainly concern the definition of the product data base. It is not easy to define a common database model to which all product data can be stored. The STEP standard [ISO 1994] represents an attempt do define such a common model. Although the STEP standard is gaining popularity, the currently available product data model systems are quite rare and incomplete.

The advantages of PDM systems include

- Commercially available PDM systems are maturer than product data model based systems.

- PDM systems pay more attention to process related activities than product data model systems, which focus more on the storing of technical data.

- The adaptation of a PDM system to an organisation is easier, due to the fact that there will be no need to change the currently used engineering tools.

The greatest disadvantage of PDM systems is the rather poor performance of data integrity checking. In case similar information is repeated in different documents, the PDM system can not ensure that the information is uniform in all the documents.

In the subsequent section only PDM systems are treated due to the immaturity of currently available product model system.

6.3.2 PDM tools

The user functions of PDM systems (see Section 6.1) may be implemented either with separate tools or with a united PDM system. For example, the most important function from the mission-critical application point of view - data vault and document management - can be implemented separately with a configuration management tool or with an actual PDM tool.

Within this study, there were no resources for purchasing the tools and making experiments with these in actual conditions. Instead, two reports containing comparisons of different tools were obtained.

The first report, *PDM Buyer’s Guide* [Miller et al. 1997], comprises an introduction to PDM systems, PDM product markets and evaluating PDM products as well as reviews of 52 different PDM systems.

The second report, *Configuration management tools: a detailed evaluation* [Ingram et al. 1995], contains an introduction to version control and configuration management systems and reviews of the most widespread systems.

7. Conclusions

We presented an outline for a set of object- and component-oriented software development activities for mission-critical software using the on-board control software of X-ray spectrometers as an example. The developed approach emphasises the use of software architectures, design patterns and application frameworks for real-time embedded software.

We proposed an evaluation framework for modern real-time software methodologies such as ObjectGEODE, SDT and ObjecTime. The framework included a hierarchical set of criteria that emphasised the development needs of mission-critical software. ObjectGEODE, SDT and ObjecTime were evaluated by the Space2000/sw project using the framework. We presented and analysed the results of the evaluations. In addition, product data management (PDM) solutions for mission-critical applications were studied.

The evaluated methodologies proved sound, mature and powerful real-time software development environments. Most of the essential aspects of mission-critical software are supported by the methodologies. There are no great differences in the capabilities of the methodologies.

The methodologies fail to fulfil some of the essential requirements of mission-critical software. The shortcomings of the CASE tools associated with the methodologies included a poor efficiency and quality of the generated target code. In addition, the support for complex data type and algorithm descriptions was found to be unsatisfactory, as well as that for performance evaluations and component-oriented development.

Due to their weaknesses the methodologies can be used only to a limited degree in the designing and implementation of mission-critical software. However, the benefits offered by simulation and validation of design models are convincing. Hence, the methodologies are worth considering, even if the final coding or module testing has to be done manually.

Related future development activities with the research problems involved in these are defined in the following:

1. Application of the results of this work in full-scale small mission-critical application domains completely,

2. application of the results in large application domains, and

3. elaborating the results into a systematic approach for reuse-driven development and continuous improvement of mission-critical systems.

The first step of the first category of problems includes:

- The selection of a well-understood and quite static application domain. This domain may be a clearly defined part of a larger application domain.

- The determination of the software development methods that are to be adopted.

- The adaptation and application of the proposed CASE tool evaluation framework and the selection of a tool,

- The adaptation and application of the proposed component-oriented development activities in the selected application domain,

- The application of the proposed product data management practices in the selected application domain.

The gained experience will be used to refine the proposed framework and process. New application domains may be selected for study based on the experience.

The second category of problems may include the application of the proposed CASE tool evaluation framework and development activities for new categories of application domains, such as those within the context of large telecommunication applications. The adaptation of the proposed CASE tool evaluation framework may prove straightforward thanks to its quite general nature. Telecommunications companies may be providing opportunities for a thorough CASE tool evaluation. The proposed development activities may require quite a lot of adapting and supplementing for large application domains.

The third category of problems may involve an assessment framework for software methods and tools, and one for product data management, as well as a systematic approach to reuse-driven development and an integration of these items into the continuous improvement practices of the organisation concerned.

References

Awad, M., Kuusela, J. & Ziegler, J. 1996. Object-Oriented Technology for Real-Time Systems: A Practical Approach Using OMT and FUSION. New Jersey: Prentice-Hall Inc. 276 p.

Braek, R. & Haugen, ∅. 1993. Engineering Real Time Systems. Hemel Hempstead: Prentice Hall. 0-13-034448-6.

Buschman, F., Meunier, R., Rohnert, H., Sommerlad, P. & Stal, M. 1996. Pattern-Oriented Software Architecture, a System of Patterns. Chichester, England: John Wiley & Sons. 457 p.

CIMdata 1996. Product Data Management: The Definition. An Introduction to Concepts, Benefits, and Technology. Fourth ed.. Ann Arbor, MI: CIMdata, Inc.

Douglass, B. 1998. Real-Time UML, Developing Efficient Objects for Embedded Systems. Reading, Massachusetts: Addison-Wesley. 365 p.

Ellison, K. 1994. Developing Real-Time Embedded Software in a Market-Driven Company. New York: John Wiley & Sons. 351 p. ISBN 0-471-59459-8

ESA 1991. ESA Software Engineering Standards, ESA PSS-05-0, Issue 2. ESA Publication Division, ESTEC, Noordwijk, the Netherlands, 91 p.

Fayad, M. & Schmidt, D. 1997. Communications of the ACM, Vol. 40, No. 10, pp. 32-38.

Firth, R., Mosley, V., Pethia, R., Roberts, L. & Wood, W. 1987. A Guide to the Classification and Assessment of Software Engineering Tools. Technical Report CMU/SEI-87-TR-10. 58 p.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. 1995. Design patterns: Elements of Reusable Object-Oriented Software, New York: Addison-Wesley. 395 p.

Geppert, B. & Rössler, F. 1996. Pattern-based configuring of a customized resource reservation protocol with SDL. Kaiserslautern, Germany: Computer Science Department, University of Kaiserslautern. 49 p.

Geppert, B., Rössler, F., Feldmann, R. & Vorwieger, S. 1998. Combining SDL patterns with continuous quality improvement: An experience base tailored to SDL patterns.

Proceedings of the 1st Workshop of the SDL Forum Society on SDL and MSC, vol. 1. Berlin, Germany, June 29 - July 1. Berlin: Humboldt University Berlin. Pp. 97- 106.

Gomaa, H. 1993. Software Design Methods for Concurrent and Real-Time Systems. Reading, Massachusetts: Addison-Wesley. 447 p.

Halttunen, V. & Hokkanen, M. 1995. Tuotetiedonhallinta. Taustaa ja ratkaisuvaihtoehtoja. Espoo, Finland: Technical Research Centre of Finland (VTT). 75 p. (1235-0605 VTT Research Notes.) ISBN 951-38-4746-2 (In Finnish including an English abstract)

Harel, D. & Rolph, S. 1989. Modeling and Analyzing Complex Reactive Systems: The Statemate Appraoch. In proceedings of the AIAA Computers in Aerospace VII Conference, Washington, DC, October 1989. Pp. 239-246.

Hess, J., Cohen, S., Kang, K., Peterson, A.S. & Novak, W. 1990. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Pittsburgh, PA: Software Engineering Institute. 149 p. (SEI-90-21)

Ihme, T. 1997. Adaptive Scenarios and Component-Based Embedded Software. ObjectMagazine Online, July 1997.

Ihme, T. 1998a. Object-Oriented Development of X-ray Spectrometer Software. Proceedings of the Colloquium on Object Technology & System Re-Engineering (COTSR). Leicester, England: De Montfort University. 14 p.

Ihme, T. 1998b. An SDL Framework for X-ray Spectrometer Software. Proceedings of the 1st Workshop of the SDL Forum Society on SDL and MSC, vol. 1. Berlin, Germany, June 29 - July 1. Berlin: Humboldt University Berlin. Pp. 125- 134.

Ihme, T. 1998c. SDL and ROOM Frameworks for Spectrometer Controllers, Version 1.0. Oulu: VTT Electronics. 9 p. + app. 62 p. (Technical Report)

Ihme, T., Holsti, N., Kumara, P., Paakko, M. & Suihkonen, K. 1998. CASE Tool Evaluation. Proceedings of the 3rd International Conference on Data Systems in Aerospace Conference (DASIA ‘98), Athens, Greece, May 1998. 6 p. (ESA SP-422.)

Ingram, P. Burrows, C. & Wesley, I. 1995. Configuration management tools: a detailed evaluation. London: Ovum Ltd.

ISO 1994. ISO 10303. Industrial automation systems and integration -- Product data representation and exchange.

Kumara, P., Suihkonen, K., Holsti, N., Paakko, M. & Ihme, T. 1998. Evaluation of the ObjectGEODE Methodology. Proceedings of the 3rd International Conference on Data Systems in Aerospace Conference (DASIA ‘98), Athens, Greece, May 1998. 5 p. (ESA SP-422.)

Leppälä, K., Korhonen, J., Ruuska, P., Toivanen, J. & Päivike, H. 1996. Real-time approach for development of scientific space instrument software. Proceedings of the 8th Euromicro Workshop on Real-Time Systems, L'Aquila, Italy, 12 - 14 July 1996. Los Alamitos, CA: IEEE Computer Society Press. Pp. 139-144.

Miller, E. MacKrell, J. & Mendel, A. 1997. PDM Buyer’s Guide. Sixth Edition. Ann Arbor, MI: CIMdata, Inc.

OORT 1996. ObjectGEODE Method Guidelines, Version 1.0. Toulouse, France: Verilog SA. 146 p.

Paakko, M. & Holsti, N. 1997a. Modelling SIXA on-board Software with SDL. Espoo, Finland: Space Systems Finland Ltd. 27 p. (Technical Report SSF-S2000-RP-001, Issue 1)

Paakko, M. & Holsti, N. 1997b. Experience on the USE of Framework for Evaluation of CASE Tools with ObjectGEODE. Espoo, Finland: Space Systems Finland Ltd. 11 p. (Technical Report SSF-S2000-RP-002, Issue 1.0)

PDMIC 1997. Understanding Product Data Management. PDM Information Company (http://www.pdmic.com/undrstnd.html).

Robinson, P. 1992. HOOD: Hierarchical Object-Oriented Design. Hemel Hempstead: Prentice-Hall Inc. 238 p.

Salmela, M. 1994. VTT Evaluation of StP/OMT Product, IDERS Project EP8593. Oulu, Finland: VTT Electronics. 28 p. (Technical Report IDERS-VTT-15-V1.1)

Selic, B., Gullekson, G. & Ward, P. 1994. Real-Time Object-Oriented Modeling. New York: John Wiley & Sons. 525 p.

SOMT 1996. SDT 3.1 Methodology Guidelines Part1: The SOMT Method. Malmö, Sweden: Telelogic AB. 196 p.

Suihkonen K. 1998. Application of an object-oriented method in Electrical Ground Support Equipment. Tampere, Finland: Tampere University of Technology, Department of Information Technology. 58 p. (In Finnish including an English abstract)

Suihkonen, K. & Kumara, P. 1997a. Answers to Specific Questions for Evaluation of CASE Tools. Tampere, Finland: Patria Finavitec Oy Systems. 20 p. (Technical Report 635-4000-RP-002, Issue 1)

Suihkonen, K. & Kumara, P. 1997b. CASE Tool Evaluation, Final Report. Tampere, Finland: Patria Finavitec Oy Systems. 17 p. (Technical Report 635-4000-RP-004, Issue 1)

Suihkonen, K. & Kumara, P. 1997c. Answers to Evaluation Questions for CASE Tools, Tampere, Finland: Patria Finavitec Oy Systems. 25 p. (Technical Report 635-4000-RP-003, Issue 1)

Toivanen, J. 1995. SIXA On Board Software Requirements Document, Version 3.0. Oulu: VTT Electronics. 75 p. (Technical Report SIXA-FM-05)

Toivanen, J. 1998a. SIXA Modelling Experimentation with the Telelogic SDT Tool, Version 1.0. Oulu: VTT Electronics. 17 p + app.19 p. (Technical Report)

Toivanen, J. 1998b. Product Data Management and Space Applications, Versio 1.0. Oulu: VTT Electronics. 20 p. (Technical Report)

UML 1997. UML Summary, Version 1.1. Santa Clara, California: Rational Software Corporation.19 p.

Vierimaa, M., Taramaa, J., Puustinen, H., Suominen, K. & Ketola, T. 1998. Framework for Tool Evaluation for a Maintenance Environment. Journal of Software Maintenance: Research and Practice, Vol 10, no.3. Pp. 203-224.

Wood, B., Pethia, R., Gold, L. & Firth, R. 1988. A guide to the assessment of software development methods. Technical Report CMU/SEI-88-TR-8. 52 p.

Appendix A: A Framework for the Evaluation of CASE Tools

CASE tools play an essential role in the software development of embedded software. However, the tool selection should be carried out only after the determination of the software development methods to be adopted, because the tools are based on the methods and provide automated assistance for designers using these methods. A CASE tool assessment process may involve the following four steps [Firth et al. 1987]:

1. Analyse the purposes for which the tool will be used,

2. Analyse the environment in which the tool will be used,

3. Develop a list of candidate tools that may meet above needs, and

4. Apply a set of assessment criteria to each of the candidate tools and select a tool.

This document focuses on the fourth step of the tool assessment process. This step consists of the following four phases [Firth et al. 1987]:

1. Establish evaluative criteria,

2. Define specific experiment,

3. Execute the experiment, and

4. Analyse the results.

The evaluation framework described by this document has been tried and improved through several experiments [Ihme et al. 1998, Paakko & Holsti 1997, Salmela 1994, Suihkonen & Kumara 1997].

Establishing evaluative criteria

A CASE tool evaluation assesses how well the tool does its job relative to the needs of the evaluator. The evaluations are subjective since every organisation has different requirements, products, skills, expertise, number of designers, software environments, methods, practices, and ideas about how tools ought to work. However, a great number of the evaluation questions that the evaluator wishes to asks about the tool can be standardised, while also accepting that different evaluators will augment questions and interpret the answers in different ways, and attach different degrees of importance to the questions and answers.

Evaluation criteria for a broad range of CASE Tools

The evaluation criteria used in this framework have been adapted from [Firth et al. 1987], [Salmela 1994] and [Vierimaa et al. 1998]. This hierarchical framework comprises the following six high level criteria [Firth et al. 1987], each including a set of lower level criteria:

1. Ease of Use
    - Tailoring
    - Intelligence & Helpfulness
    - Predictability
    - Error Handling
    - Multiple Users
    - System Interface

2. Power
    - Tool Understanding
    - Tool Leverage
    - Tool State

3. Robustness
    - Consistency
    - Evolution
    - Fault Tolerance
    - Self Instrumentedness

4. Functionality
    - Methodological Support
    - Correctness

5. Ease of Insertion
    - Learnability
    - SW Engineering Environment

6. Quality of Commercial Support

- Tool History
- Vendor History
- Purchase, Licensing, Rental
- Maintenance Agreement
- User's Groups & Feedback
- Installation
- Training
- Documentation

The first four groups mainly concern the actual CASE tool users. The last two groups concern managers of the software development environment. The criteria apply across a broad range of CASE tools.

Each low level criterion includes of a set of evaluation questions that are presented in the Appendix B. The Microsoft Excel tables for specifying a numerical credit for each question are presented in [Ihme 1998]. A scale number is used for defining the maximum numerical credit for each question.

A maximum score is defined for each low level criterion. The score of each low level criterion is calculated using the following formula:

$$\text{Score}_L = \text{MaximumScore}_L * (\Sigma\ \text{CreditsOfQuestions}) / (\Sigma\ \text{ScalesOfQuestions})$$

The score of each high level criterion equals the total of the given scores of the associated low level criteria. The total of the maximum scores of the associated low level criteria equals the maximum score of the high level criterion.

Capability profiles of the tools

The capability profiles of the evaluated tools are expressed as a percentage of the given score to the maximum score of each criterion:

$$\text{Profile}_L = (\text{Score}_L / \text{MaximumScore}_L) * 100\ \%$$

The percentages of the capability profiles may be converted to the following four textual credits of the criteria using the following rules:

Textual Credit	Percentage
Not satisfied	0 – 39.9
Partially satisfied	40 - 59.9
Largely satisfied	60 – 79.9
Fully satisfied	80 - 100

The textual credits are defined as follows:

- “Not satisfied” indicates that the tool does not have this function or property or the tool only very weakly supports the function or property and the criterion is not satisfied.
- "Partially satisfied" indicates that the tool supports the function or property in question weakly and the criterion is partially satisfied.
- "Largely satisfied" indicates that the tool supports this function or property significantly and the criterion is largely satisfied.
- "Fully satisfied" indicates that the tool supports this function or property completely or almost completely and the criterion is fully satisfied.

The textual credits are used for interpreting the numeric capability profiles.

Adapting the evaluation criteria

Each individual organisation can make its own estimate on the relative importance of each higher or lower level criterion or question. The maximum scores of the high and low level criteria can be modified when necessary. The degree of importance for each question can be lowered or heightened by modifying the relative weight of the question. Non-applicable questions can and should be suppressed in order to concentrate on those criteria that are the most relevant ones for the organisation in question.

It is possible for each organisation to specify a set of mandatory requirements for desired criteria or questions whose importance could not be expressed through credits or scores only. The evaluator can specify mandatory requirements for CASE tools using the following categories for example:

- Minimum credits for some evaluation questions

- Minimum scores for a couple of the high or low level criteria.
- Mandatory requirements such as the availability of a PC platform,
- Selectively mandatory requirements specify parts that are selectively mandatory. For example, there may be two version control systems that are selectively mandatory.
- Bound Objects requirements indicate that specified objects are bound up with one another. For example, a Unix platform and a software configuration management system may be specified as bound objects.

Providing complete answers to the evaluation questions calls for extensive background knowledge and a quite lot of specific experimenting and testing with the tool. The evaluation framework can thus be used as a checklist only. This does not require too much extra work.

Augmentation of the evaluation criteria

Each high level evaluation criterion for a broad range of CASE tools may be augmented with additional detailed criteria that fulfil any specific needs of the organisation. For example, the Functionality criterion may be augmented with a set of additional questions that deal with how well the tool supports a specific methodology. It is recommended that the additional specific questions are answered before the general functionality questions.

Defining specific experiments

Many of the evaluation questions result in tests that must be performed on each of the candidate tools. Many specific tests have to be tailored to the individual tools.

Executing experiments

Many of the evaluation questions can be answered through a review of product manuals and literature. However, the manuals can sometimes be misleading or misinterpreted. Many tests have to be performed using representative case examples for the embedded software developed by the organisation in case.

Analysing results

The collected evaluation data and tool capability profiles should be carefully analysed before making any final tool choices. The results usually indicate that no tool will provide a perfect match for all the needs and requirements of a particular organisation. The highest capability score does not always indicate the best tool choice for the organisation. It is important to pay enough attention also to those essential criteria that showed low credits or scores in the evaluation. Although the tool evaluation criteria facilitates the tool selection process, it cannot provide any absolute recipe for choosing the most appropriate and useful tool for the specific purposes of the target organisation.

References

Firth, R., Mosley, V., Pethia, R., Roberts, L. & Wood, W. 1987. A Guide to the Classification and Assessment of Software Engineering Tools. Technical Report CMU/SEI-87-TR-10. 58 p.

Ihme, T. 1998. A Framework for the Evaluation of CASE Tools, Version 1.0. Oulu: VTT Electronics. 6 p. + app. 23 p. (Technical Report)

Ihme, T., Holsti, N., Kumara, P., Paakko, M. & Suihkonen, K. 1998. CASE Tool Evaluation. Proceedings of the 3rd International Conference on Data Systems in Aerospace Conference (DASIA ‘98), Athens, Greece, May 1998. 6 p. (ESA SP-422.)

Paakko, M. & Holsti, N. 1997. Experience on the USE of Framework for Evaluation of CASE Tools with ObjectGEODE. Espoo, Finland: Space Systems Finland Ltd. 11 p. (Technical Report SSF-S2000-RP-002, Issue 1.0)

Salmela, M. 1994. VTT Evaluation of StP/OMT Product, IDERS Project EP8593. Oulu, Finland: VTT Electronics. 28 p. (Technical Report IDERS-VTT-15-V1.1)

Suihkonen, K. & Kumara, P. 1997. CASE Tool Evaluation, Final Report. Tampere, Finland: Patria Finavitec Oy Systems. 17 p. (Technical Report 635-4000-RP-004, Issue 1)

Vierimaa, M., Taramaa, J., Puustinen, H., Suominen, K. & Ketola, T. 1998. Framework for Tool Evaluation for a Maintenance Environment. Journal of Software Maintenance: Research and Practice, No.10. Pp. 203-224.

Appendix B: Evaluation Questions for CASE Tools

The evaluation criteria in this CASE tool evaluation form are hierarchical and have the following six high level criteria that include a set of lower level criteria:

1. Ease of Use
2. Power
3. Robustness
4. Functionality
5. Ease of Insertion
6. Quality of Commercial Support

The first four groups mainly concern actual users of the tool. The last two groups concern managers of the software development environment. Each lower level criterion includes of a set of evaluation questions. Answers to the questions may be given in writing directly after each question. A credit for the property or requirement evaluated by each question may be presented in tables in [Ihme 1997].

The evaluation questions have been tried and improved through several experiments [Paakko & Holsti 1997, Salmela 1994, Suihkonen & Kumara 1997a].

Ease of use

The benefits of a tool must offset its cost and the time spent using it.

Tailoring

The answers to the tailoring questions indicate how well the tools can be tailored to the varying needs of a wide range of organisations and users.

1. Can various aspects of the interface be tailored to suit user needs, including application and ability level?
2. Can the user define new commands or macros for commonly used command sequences? Do the macros allow chaining together?
3. Can the user turn off unwanted functions that might be obtrusive?
4. Can the input and output formats of the tool be redefined by the user?
5. Can the tailoring operations be controlled to maintain consistency within the user’s project or organisation?

6. Can the tool be configured by the user for different resource trade-offs to optimise such things as response speed, disk storage space and memory use?
7. Does the vendor support and assist tailoring the tool to specific user's needs?

Intelligence and Helpfulness

The tool should help the user, anticipate user interaction and provide simple and efficient means of executing the functions required by the user.

1. How well does the tool conform to GUI standards with advanced usage, e.g. is it easy to find or locate menu functions?
2. Does the tool propose well-chosen default values?
3. Can the tool be used without comprehensive training?
4. How well does the tool perform in providing on-line help, e.g. does the tool always offer context-sensitive on-line help?
5. Can novice users instantly start using the tool?
6. How well does the tool support expert users, e.g. can expert users perform special routines for running tasks more efficiently?
7. Is the tool interactive, e.g. does it prompt for command parameters, complete command strings, check for command errors?
8. Is action initiation and control left with the user when it is necessary?
9. Is quick and meaningful feedback on system status and progress of interaction and execution given to the user?
10. Can the user access and retrieve stored information quickly and with little effort whilst using the system?
11. Can the user navigate (browse) easily among hierarchical diagrams and model structures?

Predictability

Unpredicted responses are likely to result in unwanted output and unhappy users.

1. Are the responses from the tool expected in most cases?
2. Is it possible to predict the response of the tool for different types of error conditions?

Error Handling

The tool should tolerate user errors and it should be able to check for and correct these errors whenever possible.

1. Does the tool recover from errors easily?

2. Does the tool protect the user from costly errors?
3. Does the tool periodically save intermediate objects?
4. Does the tool offer protection against damage to its database caused by inadvertent tool execution?
5. Does the tool help the user with the error correction?
    - By indicating the reason for the error?
    - By indicating the location of the error?
    - By suggesting a correction?
6. Will the tool check for application-specific errors, such as checking if parentheses match?

Multiple Users

The multiple users feature denotes that a tool can be used by many users in a project.

1. Is the tool designed to be used by more than one person at a time?
2. Can several users work on different subparts of the same system at the same time?
3. Is the licensing floating (no physical devices) and does it limit the number of concurrent users instead of named users?
4. Does the tool provide for management, including access control, of work products for single and multiple users?

System Interface

The system interface feature denotes the capability of interfacing with other tools, important data formats or devices.

1. Is the interface compatible and consistent with other tools in the tool set or with other commercially available tools?
2. Can data be imported into the tool in the most important formats?
3. Can data be exported from the tool in the most important formats?
4. Does the tool provide for output devices such as printers?
    - Printing must be flexible so that a user can recursively print the whole system or its subparts with a single command
    - Printing must be possible to the printer and into a file

Power

The power of a tool depends upon several factors such as:

- the extend to which the tool “understands” the objects it is manipulating,

- the extend to which simple commands can cause major effects,
- ability of the tool to keep knowledge about its internal state, and
- the efficiency of the use of the computing resource.

Tool Understanding

The characteristic of tool understanding stands for the capability of a tool to manipulate objects that have an inner structure. It is important for the tool to understand the object content and structure and to be able to handle different aspects of that structure.

1. Does the tool operate on objects at different levels of abstraction or at different levels of detail?
2. Can the tool provide different views of diagrams and attach them to the documentation?
3. Can the tool modify collections of objects so as to preserve relationships between them?
4. Can the tool remove objects and repair the structure and insert objects with proper changes to the new structure?
5. Does the tool perform any validation of objects or structures?
    - On-fly automatically?
    - Allowing to run exhaustive consistency and logical checking of the (sub)system?
6. Can the tool create structural templates automatically?
7. Can the tool expand objects into templates of the next level of detail, with consistency checking between levels?

Tool Leverage

The leverage of the tool denotes the extent to which small actions or commands by the user can produce extensive effects.

1. Can commands be bound to specific object types or structure templates?
2. Can commands be applied systematically to entire collections of similar objects?
3. Can polymorphic commands be applied to entire structures that contain diverse objects?
4. Can commands be executed indefinitely (recursively) until a predicate is satisfied?

Tool State

If the tool remembers how it has been used in a current session or in previous sessions, it can provide the user with simple ways of enhancing the power of the tool.

1. Does the tool keep a command history?
2. Can commands be reinvoked from the history?
3. Can the command history be saved to be replayed by a new tool run?
4. Can the reinvoked commands be modified when they are replayed?
5. Can one save the current state of the tool and the objects it is manipulating, and subsequently restore these?
6. Does the tool learn, i.e. does it keep state across invocations?
7. Does the tool keep or employ statistics of command frequency and operand frequency?

Performance

The performance of a tool affects the ease of use, thus it may determine the success of the tool within an organisation.

1. Is the response of the tool to simple or frequently used commands acceptable relative to the complexity of the operations performed by the command? E.g. does the tool keep the user waiting for unreasonable long periods, or is there any response lag on commands?
2. Is the response of the tool to infrequently used commands acceptable relative to the complexity of the operations performed by the command? E.g. does the tool keep the user waiting for unreasonable long periods when the tool is generating target code?
3. If the tool supports multiple users, is the response and command execution time acceptable under maximum load?
4. Can the tool, running on the user's hardware, handle a development task of the size required by the user?
5. Does the tool provide a mechanism for disposing of any useless by-products (e.g. intermediate files) it generates?

Robustness

The robustness of a tool is a combination of several factors such as

- the reliability of the tool,
- the performance of the tool under failure conditions,
- the criticality of the consequences of tool failures,
- the consistency of the tool operations, and
- the way in which the tool is integrated into the environment.

Consistency

The following questions are concerned with the consistency of the tool operation.

1. Does the tool have standard or well-defined syntax and semantics for supported languages and models?
2. Can the output of the tool be archived and selectively retrieved and accessed?
3. Can the tool operate in a system with an unique identification for each object?
4. Can the tool re-derive a deleted unique object, or does the re-derivation create a new unique object?
5. Does the tool have a strategy for dealing with re-derivation of existing objects, involving finding the objects rather than re-deriving them?

Evolution

A tool evolves to accommodate itself to changing requirements, changes in the environment, corrections of detected flaws, and performance enhancements.

1. Is the tool built in such a way that it can evolve and retain compatibility between versions?
2. Can the tool accommodate changes to the environment in which it operates smoothly?
3. Can new versions of the tool interface with old versions of other related tools?
4. Can new versions of the tool operate correctly on old versions of target objects?
5. Can old versions of the tool operate correctly on new versions of target objects?
6. Can separate versions of the tool coexist operationally on the system?
7. Has the tool been implemented on or ported to various hosts?
8. Can the tool output be interchanged between hosts?

Fault Tolerance

The following questions are concerned with fault tolerance specifically related to the tool.

1. Does the tool always exhibit predictable behaviour and no crashes even under heavy use?
2. Does the tool give error messages and context sensitive help, or does the tool crash in case of user misuse or invalid data?
3. Does the tool show a well-defined atomicity of action so that any environmental failures during the execution of the tool do not cause irreparable damages?
4. If the system or tool is found to operate incorrectly, can the system be rolled back (reversed) to remove the effects of the incorrect actions? E.g. in case of a system

error, does the tool save the information automatically, and suggest the most recent not-defective version to be used after recovery?

Self Instrumentedness

A tool should be self-instrumented to be able to assist in determining the cause of a problem or a bug once symptoms are detected.

1. Does the tool contain any instrumentation to allow for easy debugging?
2. Are there any tools for analysing the results collected by the instrumentation?
3. Does the tool contain any self-test mechanism to ensure that it is working properly?
4. Does the tool record, maintain, or employ failure records?

Functionality

The functionality of a tool is determined by the tasks that the tool has been designed to perform and by the methods or methodologies used to accomplish the tasks. Functionality is the most important criterion for the CASE tool selection. The questions presented here deal with how well the tool automates and supports the required methodologies in general. An example set of the questions dealing with how well a tool supports a particular methodology are presented in [Suihkonen & Kumara 1997b]. It is recommended that the specific questions will be answered before the general functionality questions.

Methodological Support

The following questions deal with how well the tool automates and supports associated methodologies.

1. Does the tool support all required methodologies?
2. Does the tool provide means for integrating other methodologies?
3. Does the tool support all the required aspects of the supported methodology or methodologies?
    - Does the tool support all the aspects of the supported methodology?
    - If some aspects are excluded, are the important parts or concepts of the methodology supported? (parts that are either important to the methodology itself or important to the development project)
4. Does the tool support the communication mechanisms of the methodology (such as a textual or graphical language) without alteration?

5. Does the tool offer any other useful functions, in addition to the direct support of the methodology?
6. Is the tool free of functions that are useless or provide hindrances?
7. Does the tool support the methodology flexibly? For example, can the user initially skip or exclude some parts of the methodology and return to it later?
8. Does the tool provide an adequate scheme for storing, organising and manipulating the products of the methodology application?
9. Does the tool provide guidance to ensure that the concepts of the methodology are followed when using the tool?
10. Does the tool support syntactic and semantic checking to ensure that the concepts of the methodology are followed when using the tool?

Correctness

A tool must operate correctly and produce correct outputs.

1. Does the tool generate an output that is consistent with what is dictated by the methodology?
2. Does the tool check the methodology, if it is being executed correctly?
3. Are there any cases in which data items entered by the user are unintentionally or unexpectedly altered by the tool?
4. Is the executable output generated by the tool ‘‘bug free”?
5. Is the output generated by the tool correct by all standards?
6. Do the transformations executed by the tool always generate correct results?

Ease of Insertion

One of the important aspects of a tool use is the ease with which it can be incorporated into the target organisation and environment.

Learnability

Learning how to use the tool can result in considerable expenses and take up a lot of time.

1. Is the complexity of the tool proportional to the complexity of the application; i.e. does the tool simplify the problem rather than complicate it?
2. Do prospective tool users have the background information and experience necessary for using the tool successfully?
3. Can the users use the tool without memorising an inordinate number of commands?

4. Do the interactive elements imply corresponding functions in the problem domain, i.e. do the command names suggest functions or are the used graphical symbols representative of the related functions?
5. Are the commands and command sequences consistent throughout the system?
6. Can the user do something quickly to see what is happening and evaluate the results without a long set-up process?
7. Can the results, i.e. the work products produced, of learning exercises be disposed of easily? For example, can they be removed from a database without the involvement of a database administrator?
8. Does the tool rely on a small number of easy-to-understand/learn concepts that are explained clearly?
9. Does the tool provide a small number of functions (e.g. commands and directives) that allow the user to easily carry out the work the tool is intended for?
10. Can the user learn a small number of simple commands initially, and gradually add more advanced commands as proficiency is developed?
11. Does the tool provide the user with templates or other assistants to help with the interaction?
12. Is there a method of using a help facility that will help the novice user by providing a step-by-step description of what to do?
13. Is the time required for understanding and becoming proficient in using the tool acceptable for:
    - The average user?
    - The average project manager?
    - The project team?

Software Engineering Environment

A tool must fit the software engineering environment in which it will be used.

1. Is the tool in some ways similar to what the organisation currently does and knows? For example, is there any resemblance in the underlying methods, processes, vocabulary, notations, etc.?
2. Is the command set free of conflict with the command set of other tools the organisation uses, i.e. involving common or similar commands with different actions?
3. Does the tool run on the hardware/operating system (OS) the organisation currently uses?
4. Is installing the tool a simple, straightforward process?
5. Does the tool use file structures/databases similar to those currently in use?
6. Can data be interchanged between the tool and other tools currently employed by the organisation?

7. Can the tool be supported cost-efficiently by those responsible for maintaining the environment?
8. Can the vendor name other vendors or tools which may complement the tool set?

Quality of Support

A tool without adequate commercial support may become useless very quickly.

Tool History

1. Does the tool have a history that indicates it is sound and mature?
2. Has the tool been applied in a relevant application domain?
3. Is a (complete) list of (all) users that have purchased the tool available?
4. Is it possible to obtain any evaluations of the tools from a group of users?

Vendor History

1. Is there a staff assigned to user support?
2. Does any of your acquaintances or colleagues have any experience with dealing with the vendor? Does the vendor live up to its commitments, promises?
3. Are the projections for the future of the company positive? For example, does the company's future appear stable?

Purchase, Licensing or Rental Agreement

1. Is the contract or agreement explicit enough so that the customer knows what is or is not being acquired?
2. Is there a cost reduction for the purchase of multiple copies?
3. Is there a corporate site license available?
4. If the user wishes, can the tool be leased?
5. Does the user have the option of returning the tool for full refund during some well-defined, reasonable period of time?
6. Is the customer given full rights and access to the source code (in the event the vendor goes out of business, no longer supports the tool, and is unable to sell off the rights to the product)?
7. Is the user free of all obligations to the vendor regarding the use or sale of the objects generated by the tool (e.g. are there licence charges for run time libraries)?

Maintenance Agreement

1. Is there a warranty for the tool? (a written guarantee stating the integrity of the product and the responsibilities of the vendor regarding the repair or replacement of defective parts)
2. Can the user purchase a maintenance agreement?
3. Can the vendor be held liable for possible malfunctions of the tool?
4. Will maintenance agreements be honoured to the customer’s satisfaction in the case that the vendors sell out?
5. Is the frequency of releases and/or updates to the tool reasonable (e.g. fast enough to respond to problems, slow enough not to overburden the user with updating)?
6. Does the maintenance agreement include copies of releases/updates?
7. Is the turn-around time for problem or bug reports acceptable?

User’s Groups and User Feedback

1. Does a users’ group (or similar group that addresses the problems, enhancements, etc. with the tool) exist?
2. Does the vendor provide a responsive, helpful hot-line service?
    - automatic service, in case the tool crashes
    - telephone and telefax numbers, electronic mail and WWW for the service

Installation

1. Is the tool delivered promptly and as a complete package (including object code, documentation, installation procedure, etc.)?
2. Does the vendor provide any installation support/consultation?

Training

1. Is high-quality training available?
2. Has any prerequisite knowledge for learning and use of the tool been defined?
3. Is training customised for the acquiring organisation and individuals paying attention to the needs of different types of users (novices, advanced users, engineers, project managers, etc.)?
4. Do the training material and/or vehicles allow the user to work independently as time permits?
5. Is the user provided with examples and exercises?
6. Are the vendor representatives (marketing, sales, service, training) competent with the product and well-trained?

Documentation

1. Is the tool supported with ample documentation (e.g. installation manuals, user's manuals, maintenance manuals, interface manuals, etc.)?
2. Is any tutorial provided?
3. Quality of the documentation:
   - Does the documentation provide a description of what the tool does (''big picture view") before throwing the user into the details of how to use it?
   - Is the documentation:
     - Readable, well-structured and presented in an "easy-to-read" way?
     - Illustrative?
     - Understandable?
     - Complete and extensive?
     - Accurate and correct, without any mistakes or discrepancies?
     - Affordable?
4. Availability of the documentation
   - Is on-line help available?
   - Is the documentation available in paper format as well as in electronic format?
   - Does the documentation have an indexing scheme to help the user to find answers to specific questions?
5. Is the documentation up-to-date, and updated promptly and conveniently to reflect the changes to the implementation of the tool?

References

Ihme 1997. Tool Capability Profiles. Oulu, Finland: VTT Electronics. 3 p.

Suihkonen, K. & Kumara, P. 1997a. Answers to Evaluation Questions for CASE Tools, Tampere, Finland: Patria Finavitec Oy Systems. 25 p. (Technical Report 635-4000-RP-003, Issue 1)

Suihkonen, K. & Kumara, P. 1997b. Answers to Specific Questions for Evaluation of CASE Tools. Tampere, Finland: Patria Finavitec Oy Systems. 20 p. (Technical Report 635-4000-RP-002, Issue 1)

Paakko, M. & Holsti, N. 1997. Experience on the USE of Framework for Evaluation of CASE Tools with ObjectGEODE. Espoo, Finland: Space Systems Finland Ltd. 11 p. (Technical Report SSF-S2000-RP-002, Issue 1.0)

Salmela, M. 1994. VTT Evaluation of StP/OMT Product, IDERS Project EP8593. Oulu, Finland: VTT Electronics. 28 p. (IDERS-VTT-15-V1.1)

Appendix C: Space 2000 - Space Equipment Technology 1996-2000

The Space 2000 programme concentrates on the technology of space satellites and their ground support equipment. Although space technology covers a broad spectrum of challenging technologies, it is, however, often based on the same technological solutions as earthbound applications. Space technology offers companies and research institutes opportunities of further applying and developing their expertise and products and of extending their market to new, challenging fields of application.

Objectives

* Developing the international competitiveness of Finnish space equipment industry, especially within the satellite projects of the European Space Agency (ESA).

* Intensifying technology synergy between space technology and other industry sectors.

* Transmitting the expertise acquired in connection with the development of the nationally financed SCIENTIFIC space equipment to an industrial environment.

* Creating a functional network for space technology to enable the realisation of more extensive space technology projects in Finland.

Projects

The Space 2000 programme consists of product development projects of various companies as well as research-based joint projects of companies and research institutes. Many projects are closely related to the ESA future satellite projects and technology programmes.

Focal Areas of the Programme

* Preparation of Finnish R&D project proposals for the ESA technology programmes
* Commercialisation of space technology
* Space electronics and software
* Mechanics of space equipment
* Structural technology of satellites
* Manufacturing, testing, and quality-related activities
* Education and training in the space sector.

Duration and Budget of the Programme

The Space 2000 programme runs for five years. The total budget of the programme is estimated at FIM 50 million, of which approximately 40 percent is financed by Tekes.

International co-operation

Co-operation within the European Space Agency plays a central role in the Finnish space activities. Therefore, also the Space 2000 programme is closely linked with the ESA technology programmes and satellite projects.

For more information, please contact

Programme manager
Mr Petri Peltonen
Tekes
P.O.Box 69
FIN-00101 Helsinki, Finland
Tel. +358-105 215 855
Fax +358-105 215 900
E-mail: petri.peltonen@tekes.fi

http://www.tekes.fi/english/programm/info/spac2000/