An SDL Framework

for X-ray Spectrometer Software

Tuomas Ihme
VTT Electronics
P.O.Box 1100, FIN-90571, Oulu, Finland
telephone +358-8-551 2111, telefax +358-8-551 2320
Tuomas.Ihme@vtt.fi

Abstract

It often makes sense to invest in reusable software structures and components, if several related control systems are to be developed. This paper describes an SDL (Specification and Description Language) pattern and framework for the design of scientific on-board X-ray spectrometer control software. The framework includes reusable SDL components for a family of spectrometer controllers. The components are to be found at different hierarchy levels of SDL specifications. The SDL pattern supports the designing of a centralised control architecture of SDL models in the framework. The utilisation of general design patterns in the construction and reuse of the framework is discussed, as well as the CASE tool support and the applicability of the object-oriented features of SDL in the construction of the framework.

Keywords

SDL-92, SDL pattern, design pattern, framework, object-oriented embedded software, controller, MSC

1 INTRODUCTION

Most SDL-based life cycle approaches such as the OORT [1] and SOMT [2] 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. Object orientation in SDL provides a promising framework for making explicit reusable system structures and software components. Yet, organised reuse of SDL specifications and patterns is not a very common phenomenon.

The second section of this paper describes an SDL pattern for designing a centralised control architecture of SDL models for spectrometer control software. SDL pattern description templates [3] are utilised to describe the pattern. The presented examples are derived from the documentation of existing on-board software for multielement X-ray photon counting spectrometers.

The third section describes an SDL design framework for the Measurement subsystem of X-ray spectrometer controllers. An SDL-specific framework is defined by the TIMe method [4] 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." Actual systems may be described by defining subtypes and redefining the virtual, application-specific types in the framework. SDL packages may be used in the implementation of SDL frameworks. The packages are libraries that are used for making reusable SDL components available in different systems. The reusable components are defined as SDL types. The SDL pattern described in the second section as well as general design patterns [5,6] are used in designing the architecture of SDL models in the framework.

The fourth section describes the experience gained from the development of the SDL pattern and framework. The following commercial methodologies were used during the development of the framework: ObjectGEODE

by Verilog SA, SDT by Telelogic AB and ObjecTime by ObjecTime Limited. The ObjectGEODE methodology consists of the OORT method [1] and the ObjectGEODE Version 1.1 tool. The SDT methodology includes the SOMT method [2] and the SDT Version 3.11 tool. The ObjecTime methodology consists of the ROOM method [7] and the ObjecTime Version 4.4 tool.

2 THE MEASUREMENT CONTROL ARCHITECTURAL PATTERN

2.1 Intent

The Measurement Control architectural pattern 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 1. A single component is not often reusable on its own, but is reused in conjunction with many other related components. The MSC diagram in Figure 2 depicts a typical message scenario for the Measurement subsystem.

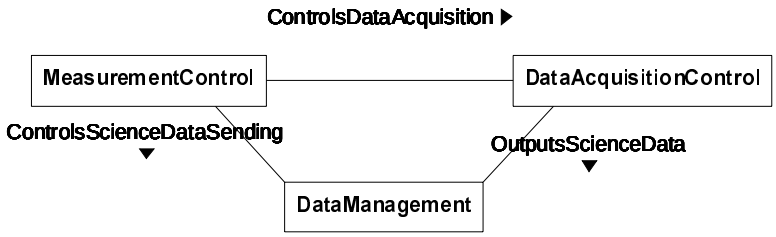

Figure 1. The structure of the measurement control architecture.

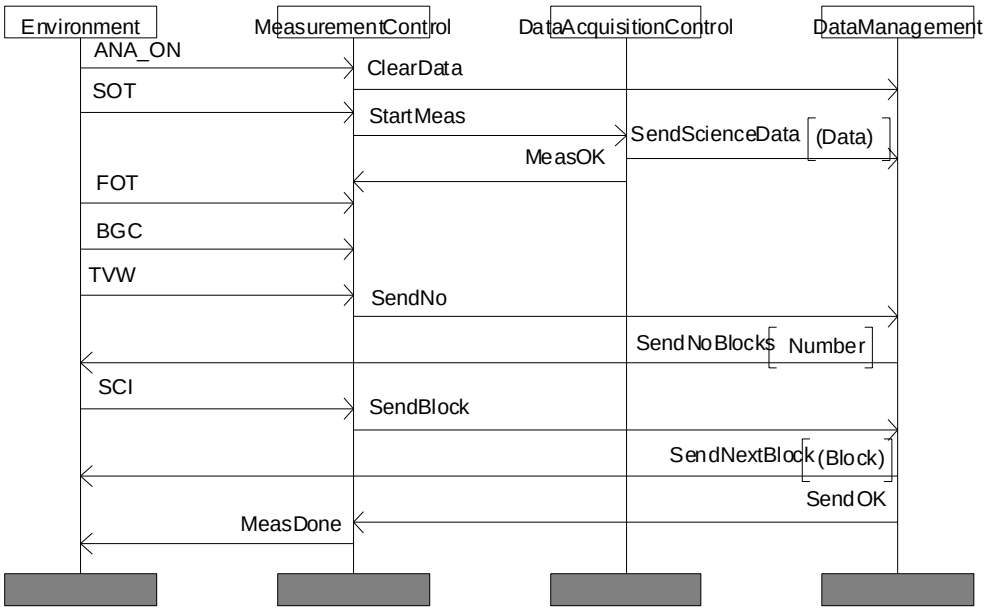

Figure 2. A message scenario typical of the Measurement subsystem.

2.2 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 to hard real-time systems requiring complete timing predictability.

2.3 Participants in the pattern

The Measurement Control class is responsible for controlling and timing the main functions of the subsystem. It provides methods to control the electronics and instances of rather 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.

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 in SDL models. 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.

2.4 Data Acquisition Control

The SDL fragment of Data Acquisition Control is shown in Figure 3. 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 the 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.

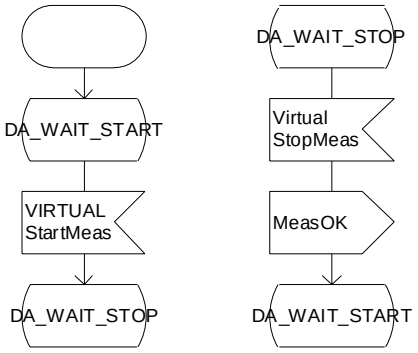

Figure 3. The SDL fragment of Data Acquisition Control.

Semantic properties:

Property 1: 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:

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. Property 1 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 several hours. The polling will have to be continuous in order to meet the requirements set on the data collection speed. The continuous polling will be replaced with periodic polling using a timer-triggered transition from the DA_WAIT_STOP state for the simulation purposes of the system.

2.5 Data Management

The SDL fragment of the Data Management component is shown in Figure 4. 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 spectrum signals from Data Acquisition Control. The state FM_WAIT may be renamed.

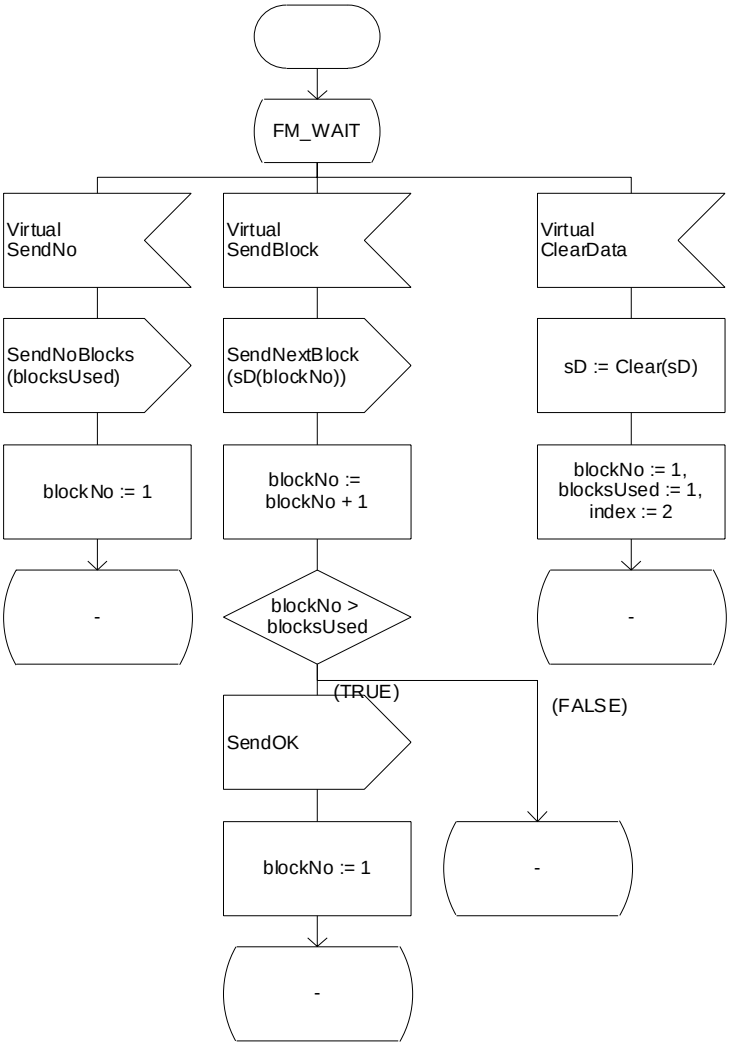

Figure 4. The SDL fragment of Data Management.

Semantic properties:
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:
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 AN SDL FRAMEWORK FOR THE SPECTROMETER CONTROLLER FAMILY

The SDL framework includes SDL models of measurement subsystems for a family of spectrometer controllers. The documentation structure of the framework is shown in Figure 5. 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 Measurement Control pattern is used in designing the control architecture of SDL models in the framework.

3.1 Interfaces for spectrometer controllers

SDL types in the Abstract Spectrometer Features package in Figure 5 define common structures and interfaces for measurement subsystems of spectrometer controllers. The SDL types are not completely defined for simulation and there are 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. The controllers differ in terms of how they operate internally. This makes use of the Strategy Pattern [5] that describes selections of different implementations of one kind of black box behaviour. The pattern is also known as Policy [6]. 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 away 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 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.

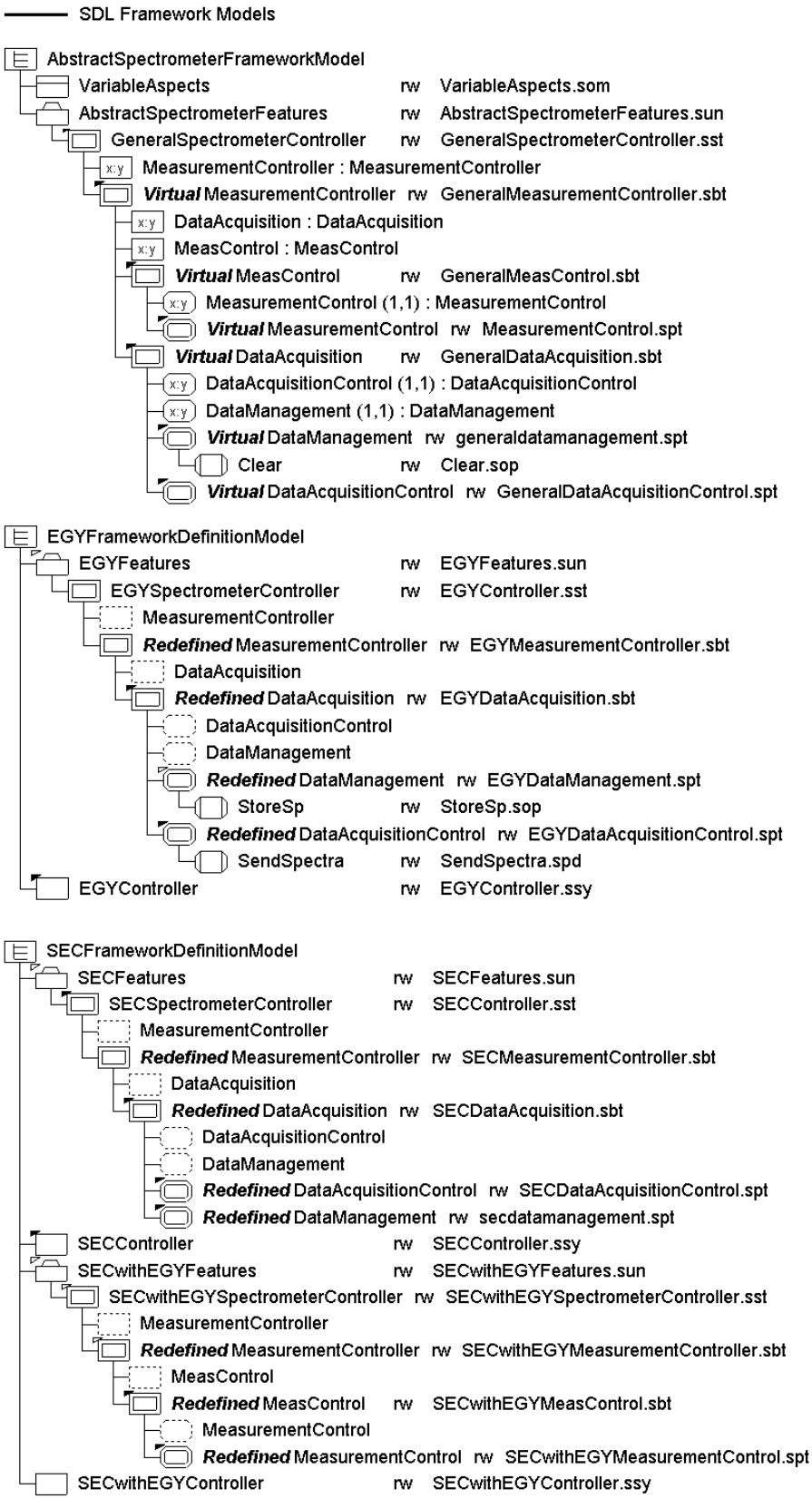

Figure 5. The documentation structure of the spectrometer controller SDL framework .

3.2 Reusability of the SDL framework

The SDL framework supports several strategies for the reuse of SDL components, for example:

- The selection of concrete SDL system models that have the same type of interface to the Ground Station (Figure 6).
- The selection of an existing system type for the designing of a new version of the Measurement subsystem. For example, the SDL model in the SECwithEGY Features package in Figure 5 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 the designing of 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 7).
- The selection of data acquisition components that have the same type of interface to the Ground Station (Figure 8).

The designer can use the strategy pattern for the selection of different Spectrometer Controller system models in shown in Figure 6. 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.

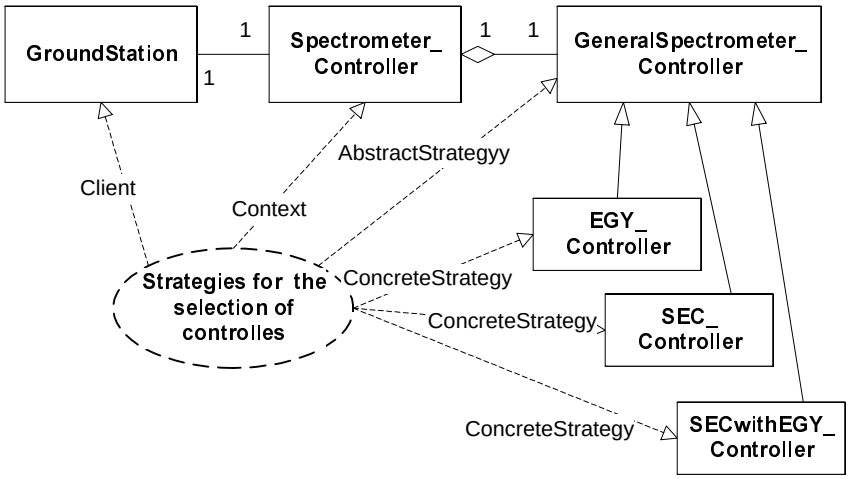

Figure 6. The use of the strategy pattern for the selection of Spectrometer Controller system models.

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 7):

- 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.

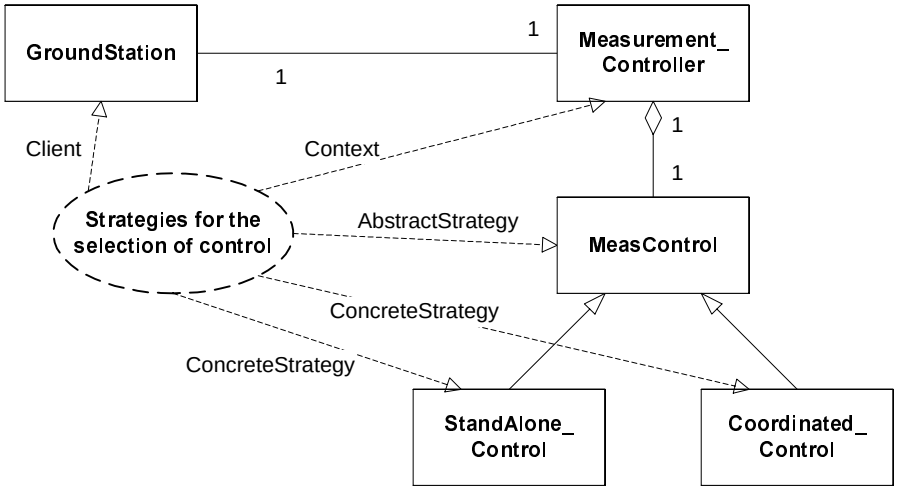

Figure 7. The use of the strategy pattern for the selection of concrete control processes.

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 8. 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 7 and 8 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 different 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.

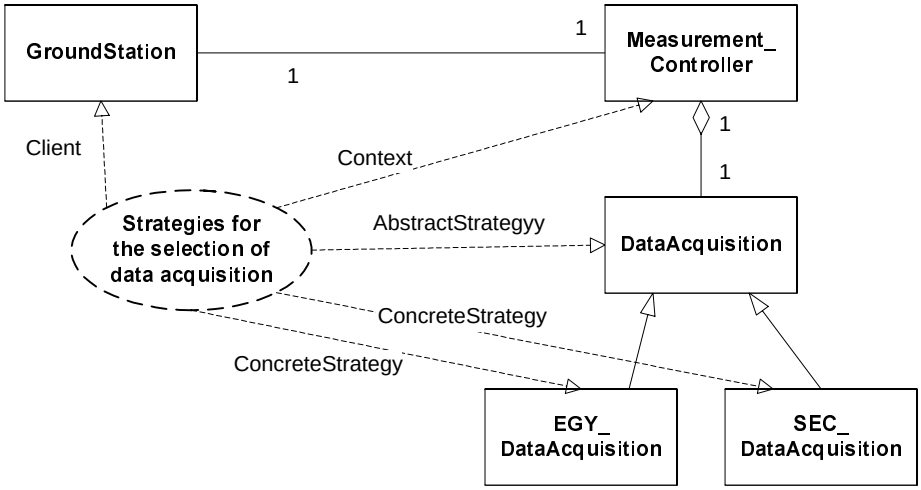

Figure 8. The use of the strategy pattern for the selection of concrete data acquisition strategies.

3.3 Documentation of the SDL 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 can 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 variability is explicit. It is very important to document the designed implicit variability of components and collections of components. The documenting 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 are documented in the Variable Aspects OMT model, which is included in the documentation structure, see Figure 5. 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.

4 DISCUSSION

The OMT and MSC notations were used in the description of structural aspects and typical interactions of the Measurement Control pattern as proposed by Geppert and Rössler [3]. The domain analysis guidelines of TIMe proved useful in the construction of the OMT structure model. The structure model of the measurement control architecture, see Figure 1, resembles the domain object model in TIMe.

The approach of Geppert and Rössler [3] 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 [3] do not have any separate descriptions.

The textual property description proposed by Geppert and Rössler [3] 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, there are shortcomings in the SDL interface support. 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. Furthermore, 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 allow using with 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. The framework proved difficult to develop, understand and maintain 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, as well as providing 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.

The disadvantages of SDL include poor efficiency of the generated code and inadequate support for complex data type, algorithm, concurrency, timing constraints, performance analysis and hardware interface descriptions. These shortcomings restrict the use of SDL for system designing and implementation. The data structures and algorithms of the Data Acquisition Control and Data Management processes have to be implemented outside SDL with the C or an assembly language. However, the validator tool will have no control over the activities and data structures implemented in C code and called from within the SDL model. Therefore, parts of the system state space will be hidden from the validator, and thus the automatic validation process will not work properly.

5 CONCLUSIONS

Examples and experiences gained in applying SDL patterns and frameworks to the designing of embedded control software for spectrometer controllers were presented. 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. The templates fail to support all the required architectural viewpoints of the Measurement Control pattern. 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 documenting of component interfaces and design, as well as the reuse strategies of the framework. The following viewpoints and problems were encountered during the development of the framework:

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

The ObjectGEODE and SDT tools fail 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 are partly based on the differences between the SDL-92 and ROOM languages.

Acknowledgements --- The work reported in this paper was funded by the Technology Development Centre of Finland (TEKES), Finnish industry and VTT Electronics.

6 REFERENCES

[1] "ObjectGEODE Method Guidelines, Version 1.0", Verilog SA, Toulouse, France, 1996.

[2] "SDT 3.1 Methodology Guidelines Part 1: The SOMT Method", Telelogic AB, Malmö, Sweden, 1996.

[3] Geppert, B.; Rössler, F.: "Pattern-based configuring of a customized resource resevation protocol with SDL", Computer Science Department, University of Kaiserslautern, Kaiserslautern, Germany, 1996.

[4] Braek, R.; Haugen, ∅: "Engineering Real Time Systems", Prentice Hall, Hemel Hempstead, 1993. 0-13-034448-6.

[5] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J: "Design patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, New York, 1995.

[6] Douglass, B.: "Real-Time UML, Developing Efficient Objects for Embedded Systems", Addison-Wesley, Reading, Massachusetts, 1998.

[7] Selic, B., Gullekson, G. & Ward, P.: "Real-time object-oriented modeling", John Wiley & Sons, New York, 1994.