Proceedings of the IASTED International Conference
APPLIED INFORMATICS
February 14-17, 2000, Innsbruck, Austria

MODELLING A REAL-TIME SYSTEM USING OBJECT-ORIENTED TECHNIQUES*

SÉRGIO F. LOPES

e-mail: sergio.lopes@dei.uminho.pt
Departamento de Electrónica Industrial
Escola de Engenharia - Universidade do Minho
Campus de Azurém
4800-058 Guimarães - PORTUGAL

JOÃO L. MONTEIRO

e-mail: joao.monteiro@dei.uminho.pt
Departamento de Electrónica Industrial
Escola de Engenharia - Universidade do Minho
Campus de Azurém
4800-058 Guimarães - PORTUGAL

ABSTRACT

Real-time systems have higher complexity and impose tougher constrains in the development process due to the different hardware support, the often distributed topology and time requirements. The development of these systems demand high quality and increasing economic constrains. These requirements point to the use of standardised specification techniques, based on hierarchical and graphical modelling. This approach have the advantage of reducing costs, time-to-market and increase of reuse of software components. Therefore the use of an integrated specification methodology has a major importance.

In the development of the present work, we have applied the Object-Oriented Real-Time Techniques (OORT) method, which is oriented towards the specification of distributed real-time systems. It is based on formal notations which are international standards. This paper, describes the method and discuss its most important aspects by applying it to the specification of the Multiple Lift System.

Keywords: Distributed Systems Specification, Real-Time Computer Control, Software Engineering, Object-Oriented.

1 INTRODUCTION

Real-time systems are very complex because they are often distributed, run in different platforms, have temporal constraints, etc. The development of these systems demand high quality and increasing economic constraints, therefore it is necessary to minimise their errors and its maintenance costs, and deliver them in short deadlines.

To achieve these goals it is necessary to verify a few conditions: decrease the complexity of the systems through hierarchical and graphical modelling for high flexibility in the maintenance; protect the investments with the application of international standards in the development; to apply early verification and validation techniques to reduce the errors; and, reduce the delivery times by automating code generation and increasing the level of reusability. Finally, its necessary to have a tool that provides these conditions. The present work was developed with the *Object*GEODE[i] toolset, that supports the OORT method.

The OORT method [13] is organised according to the diagram of figure 1 and applies the Unified Modelling Language (UML), Message Sequence Chart (MSC) and Specification and Description Language (SDL). The UML language is a de jure standard (see [12] for details) and it is defined in [10]. The MSC was defined [6] as complement to SDL, both international standards by ITU-T. [11] provides an introduction to MSC. SDL is defined by [4], [5] and [7], however [9] is a more comprehensive reference, while [3] is a handy summary of the language. The use of both languages together is guided by [8].

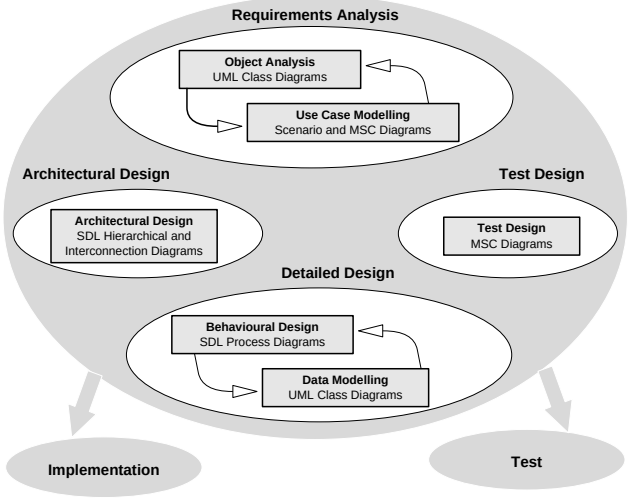

Figure 1 - The OORT method.

* This work is supported by the Fundação para a Ciência e a Tecnologia PRAXIS XXI program of the Ministério da Ciência e Tecnologia of the Portuguese Government.

In this work we have applied the OORT method to the modelling of a case study – the Multiple Lift System (MLS). A description of a MLS architecture using UML is presented in [2]. The analysis model uses UML to model the system's environment, and MSC to specify the behaviour of the system. The system's architecture is defined in SDL. The detailed design uses SDL for the concurrent objects specification and UML for the passive components description. The MSC language supports the test design activity. Each of this steps in the systems engineering process is described in the following sections.

2 REQUIREMENTS ANALYSIS

In the requirements analysis phase, the system environment is modelled and the user requirements are described. The analyst must concentrate on **what** the system should do. The environment where the system will operate is described by means of UML class diagrams – **object modelling**. The functional behaviour of the system is specified by MSCs organised in a hierarchy of scenarios **- use case modelling**.

The system is viewed from the exterior as a black box with which external entities (system actors) interact. Both the object model and use case model must be independent of the solutions chosen to implement the system.

2.1 OBJECT ANALYSIS

In the description of the system environment the class diagrams are used to express the application and services domains. This is done by identifying the relevant entities of the application domain (physical and logical), their attributes, and the relationships between them. It is also necessary, for the sake of simplicity and expressiveness, to group entities and their relationships in different modules that reflect different perspectives of the system, as is supported by [16]. Generally speaking there is one module for each of the actors that interact with the system, one for some basic system composition and other to express certain environment relationships.

In the figure 2 we present a general view of the system, where the system main actors are identified, in this case `Passenger`, `Potential Passenger` and `Operator`.

The generic system architecture is modelled in figure 3. In order to keep simple modules, each of the component classes are refined in different diagrams.

2.2 USE CASE MODELLING

The use case model is composed by the scenario hierarchy and MSC diagrams. The scenario hierarchy should contain all the different expected scenarios of interaction between the system and its environment. The goal it is to model the functional and dynamic requirements of the system. First, the main scenarios are identified, and then they are individually refined in subsequent more detailed scenarios until the terminal scenarios can be easily described by a chronological sequence of interactions between the system and its environment.

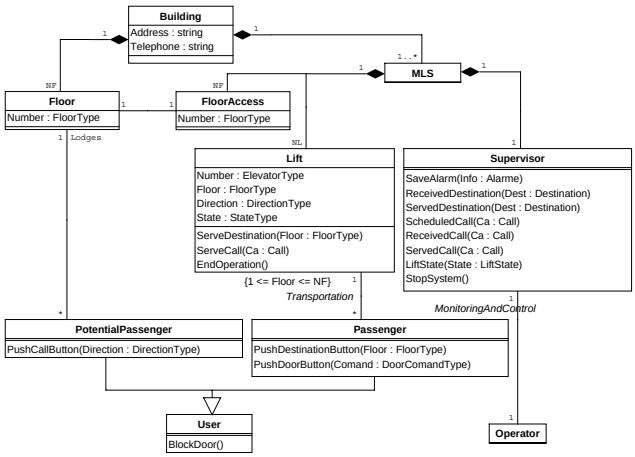

Figure 2 – Building Module UML Class Diagram.

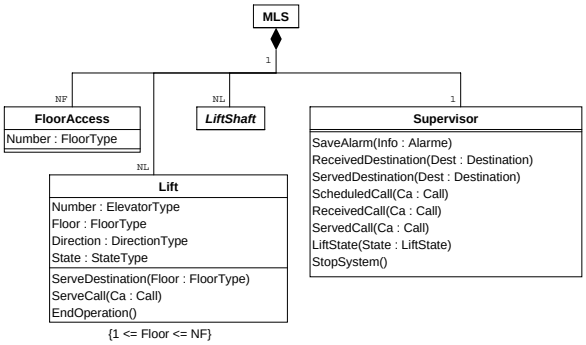

Figure 3 – System Architecture UML Class Diagram.

One problem of this approach is the scenario explosion. To deal with that difficulty we apply composition operators that combine hierarchically the several scenarios. Nevertheless, the problem is only diminished but not completely solved. It is still necessary to choose well the scenarios, namely to chose those which are the most representative of the system behaviour.

The system operation is divided in phases that are organised by composition operators, and each phase is a branch in the scenario hierarchy. Figure 4 shows the `Trip` phase scenario hierarchy, in which we have a `Floor Crossing` terminal scenario which is illustrated in figure 5.

A constant concern must be the coherence between the use case and the object models. See [13] for more details.

3 ARCHITECTURAL DESIGN

In this phase the system designers specify a logical architecture of the system (as opposed to the physical architecture). The SDL language covers all aspects of the architecture design.

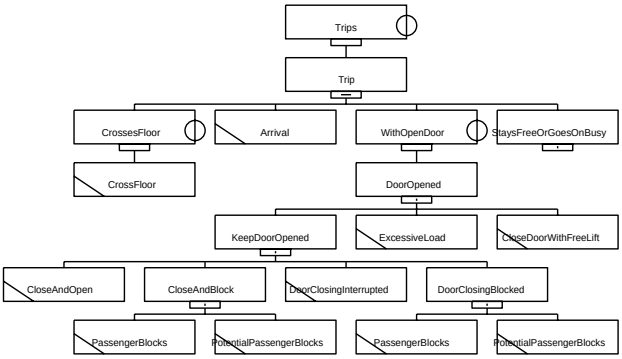

Figure 4 – Scenario Hierarchy for the Trip Sub Scenario.

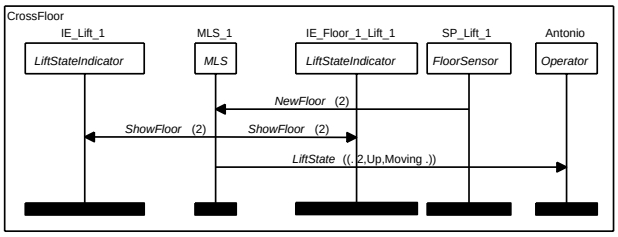

Figure 5 – Abstract MSC for the Floor Crossing Scenario.

The system is composed of **concurrent objects** (those which have an execution thread) **and passive objects** (those which implement a set of functions invoked by concurrent objects). In the architecture design phase, the concurrent objects that compose the system are identified and organised hierarchically. This is accomplished by a combination of refinement and composition. The refinement is a top-down process in which higher level objects are divided in smaller and more detailed objects, always trying to keep a good modularity. The composition is a bottom-up process in which designers try to group objects in such a way that favours reutilization and that maintains a good encapsulation of the architectural objects. Figure 6 illustrates the SDL object's hierarchy of the MLS.

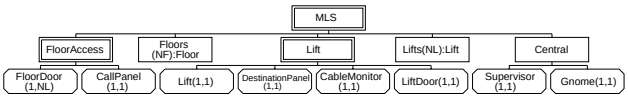

Figure 6 – MLS SDL Hierarchy Diagram.

In the architectural design, the real characteristics of the environment where the system will operate should be considered, as well as the efficiency aspects. On the other hand, the SDL model should be independent of the real object distribution on the final platform.

At the first level, the system actors are considered through their interfaces, and modelled as channels between the system top level objects and the outside world. Figure 7 shows the top level of the MLS architecture.

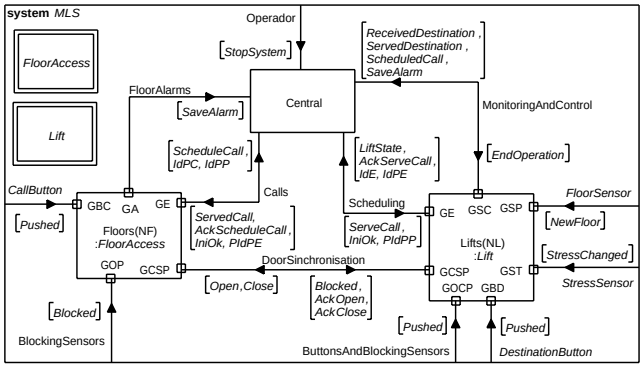

Figure 7 – SDL Interconnection Diagram of the Top Level of the MLS Hierarchy.

Some passive objects are also defined, such as signals with complex arguments, Abstract Data Types (ADTs) associated with internal signal processing, and operators to implement the I/O communication with the outside world (instead of signals).

The use of SDL assures the portability of the system architecture, since the communication service is independent of the real object distribution, the communication channels are dynamic, and the objects can be parameterised.

4 DETAILED DESIGN

The description of concurrent and passive objects that constitute the system architecture is done in the detailed design phase. In other words, it is described **how** the system implements the expected services, and it should be independent of the final platform where the system will run.

4.1 CONCURRENT OBJECTS DESIGN

The concurrent objects are the terminal objects of the SDL hierarchy. They are SDL processes and are a kind of Finite State Machine (FSM), with its states and state transitions, called **process diagrams**. The process diagrams are built by analysing the input signals of each process defined in the architecture model and how the answer to those signals depends on the previous states. The SDL has a set of mechanisms to describe the transitions that allow a complete specification of the process behaviour. In the figure 8 is shown a process diagram.

The reuse of external concurrent objects is supported by the SDL encapsulation and inheritance mechanisms.

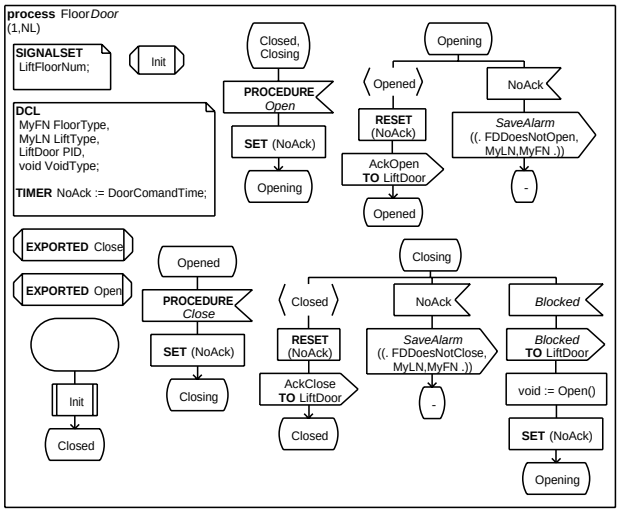

Figure 8 – SDL Process Diagram of the Floor Door Process.

4.2 PASSIVE OBJECTS DESIGN

Some passive objects are identified during the analysis phase. Generally they model data used or produced by the system, and they are included in the detailed design to provide services to concurrent objects. There are also passive objects that result from design options, such as data management, user interface or equipment interface and inclusion of other design techniques.

Although the SDL ADTs provide a way to define passive objects they are better defined by UML classes. So the ADTs from the SDL detailed design model are translated to UML classes and organised in detailed design class diagrams.

The reuse of external passive objects is facilitated by the UML encapsulation and inheritance mechanisms. These characteristics of UML, and also SDL, allow for the use of other techniques of design in certain systems. For instance, in the case of embedded systems, it can be useful to use VHDL to design some physical parts.

4.3 PORTABILITY

The multi-tasking, the communication and the time management are implemented by the SDL virtual machine, and therefore are independent of the physical platform and RTOS on which the system will run. The system maintenance is kept at the SDL specification level, thus it is easier to correct and change the system. However, the portability depends largely on the language chosen to implement the passive objects.

5 TEST DESIGN

In this phase, the communication between all the elements of the system architecture is specified by applying detailed MSCs to describe the sequences of messages exchanged between them, in all the scenarios that compose the use case model. This is done by refining the abstract MSC of each terminal scenario from the analysis according to the SDL architecture model. Consequently, the test design activity can be done in parallel with the architecture design and serve as requirements to the detailed design phase.

In the intermediate architecture levels, the detailed MSCs represent integration tests between the concurrent objects. The last step of refinement correspond to unit tests that describe the behaviour of processes (the terminal SDL architecture level). The figure 9 illustrates this.

The process level detailed MSCs can be further enriched by including in each process behaviour detailed graphical elements such as states, procedures and timers.

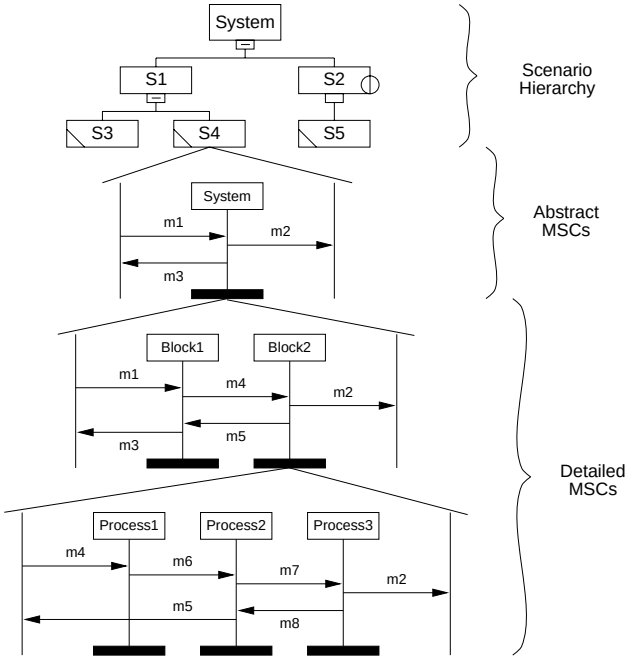

Figure 9 – Test Design in OORT.

Figure 10 shows a integration test corresponding to figure 5 abstract MSC, and figures 11 represents the respective unit test for one of the blocks.

This phase can be a very long and resource consuming, thus substantially increasing the system development cost. However, it is decisive to the system success.

The use case model reflects the user perspective of the system. The test design should be spread to cover aspects related to the architecture, such as performance, robustness, security, flexibility, etc.

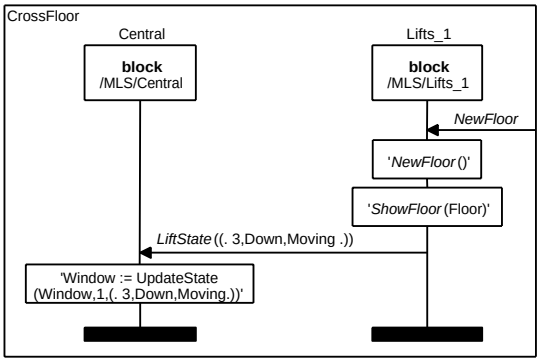

Figure 10 –Detailed MSC with Floor Arrival Integration Test.

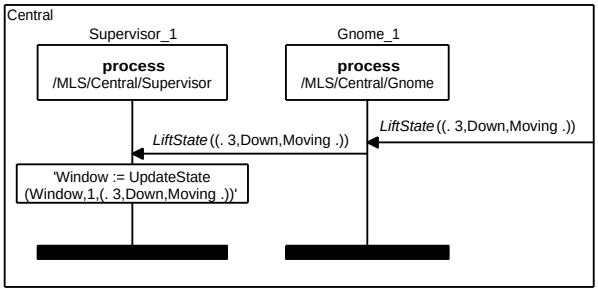

Figure 11 – Detailed MSC with Floor Arrival Unit Test of Block Piso.

6 CONCLUSION

The combined application of the languages UML, MSC and SDL here presented permits to confirm the individual, and as a whole, validity of the languages for the implementation of real-time distributed systems, because they are object-oriented and therefore they can be combined in a consistent way and simplify the reuse.

The UML class diagrams are concise and simple. The MSC and SDL are mature and well defined: they provide the essential constructions and left out mechanisms not so useful or generic. Furthermore, they are standards that are continuously being improved.

Because SDL is a formal language it can be used to define rules in the partition and synthesis of a system specification into hardware and software, as is the case of a methodology presented in [1]. Furthermore, the implementation can be automatic, thus limiting the manual coding to the non real-time operations. The generated application is scalable, because the logical architecture is independent of the physical architecture. The mapping between objects and hardware is define in the implementation phase only.

The adoption of a graphical specification methodology forces the designer to think the whole application in an organised and consistent way. Besides, formal languages permit the automatic simulation of the system [14], to make early validations and the automatic code generation. Therefore, comparing to the non formalised development, the applications are better in terms of efficiency, less errors, flexibility and easy of maintenance.

7 REFERENCES

[1] J.M. Daveau, G.F. Marchioro, T. Ben-Ismail and A.A. Jerraya, "Cosmos: An SDL Based Hardware/Software Codesign Environment", in: **Hardware/Software Co-design and Co-Verification**, eds. Bergé, J-M, Levia, O. and Rouillard, J., Kluwer Academic Publishers, 1997, 59-87.

[2] B.P. Douglass, Real-Time UML: Developing Efficient Objects for Embedded Systems (Addison-Wesley, 1998).

[3] O. Faergemand and A. Olsen, Introduction to SDL-92, *Computer Networks and ISDN Systems*, 26(9), 1994.

[4] ITU-T Recommendation Z.100, Specification and Description Languge (SDL), March 1993.

[5] ITU-T Recommendation Z.100 Appendix 1, SDL Methodology Guidelines, March 1994.

[6] ITU-T Recommendation Z.120, Message Sequence Chart (MSC), ITU, October 1996.

[7] ITU-T Recommendation Z.100 Addendum 1, Specification and Description Languge (SDL) Addendum 1, October 1996.

[8] ITU-T Recommendation Z.100 Supplement 1, SDL + Methodology: Use of MSC and SDL (with ASN.1), May 1997.

[9] A. Olsen, O. Faergemand, B. Moller-Pedersen, R. Reed, J.R.W. Smith, Systems Engineering Using SDL-92 (North Holland, 1994).

[10] OMG Unified Modeling Language Specification, Version 1.3, June 1999.

[11] E. Rudolph, P. Graubmann, J. Grabowski, Tutorial on Message Sequence Charts, *Computer Networks and ISDN Systems*, 28(12), 1996.

[12] Cris Kobryn, UML 2001: A Standardization Odyssey, *Communications of the ACM*, 42 (10), 1991, 29-37.

[13] Verilog, ObjectGEODE Method Guidelines (Verilog SA, 1996).

[14] Verilog, ObjectGEODE SDL Simulator Reference Manual (Verilog SA, 1996).

[15] Vincent Encontre, How to Use Modeling to Implement Verifiable, Scalable, and Efficient Real-Time Application Programs, *Real-Time Engineering*, Fall 1997.

[16] Yourdon E., Object-Oriented Systems Design: An Integrated Aproach (Prentice Hall, 1994).

[i] ObjectGEODE is a registered trademark by Verilog.