**Name of the workshop:** Workshop on Computer Networks

**Title of the paper:** BMS Protocol Development Process

**Abstract:**

In this paper, the protocol development process defined and regularly refined by BMS will be described. Standard languages like UML, MSC, and SDL are used to specify and describe our systems to ensure precision and traceability.

**Author:** Sherman Wang

**Affiliation:**

BENQ Mobile System Inc.

23, Li-Hsin Rd., Science-based Industrial Park, Hsinch 300, Taiwan, R.O.C.

ShermanWang@benqms.com

Tel: +886-3-6117595

Fax: +886-3-6118877

**Name of contact author:** Sherman Wang

**Keywords:** UML, SDL, MSC

# BMS Protocol Development Process

Sherman Wang

## 1 Introduction

The complete specification of a system includes various kinds of information; each of them can be presented in natural, semi-formal, or formal language. The essence of formal languages is a mathematical underpinning that can ensure precision and traceability. Three types of languages are used, UML, MSC, and SDL.

UML (Unified Modeling Language) is a standard language used in Object-Oriented Analysis and Design (OOAD). With UML, use case modeling is used to analyze the requirements, and class diagram is used to depict the software system architecture.

SDL (Specification and Description Language) is a standard language to specify and describe systems. It was developed by ITU-T and is the ITU-T Recommendation Z.100. SDL is specially concerned with the specification of behavior, structure, and data. SDL was conceived for use in telecommunication systems. With SDL, we may specify different levels of abstraction, beginning from an overview and going on to very specific details.

MSC (Message Sequence Chart) is also an ITU standard (Z.120 Recommendation). It is used to describe the interactions between sub-systems and has the similar features and representation to sequence diagram of UML.

## 2 Software Development Process

BMS has a development process team that is responsible for the defining, supervising and supporting of the development process for all products. We have adopted a spiral process model, as shown in Fig. 1, for our software development. The spiral model will reduce the project risks and encourage self-evolution in development process. It is suitable for large-scale project and junior team members.

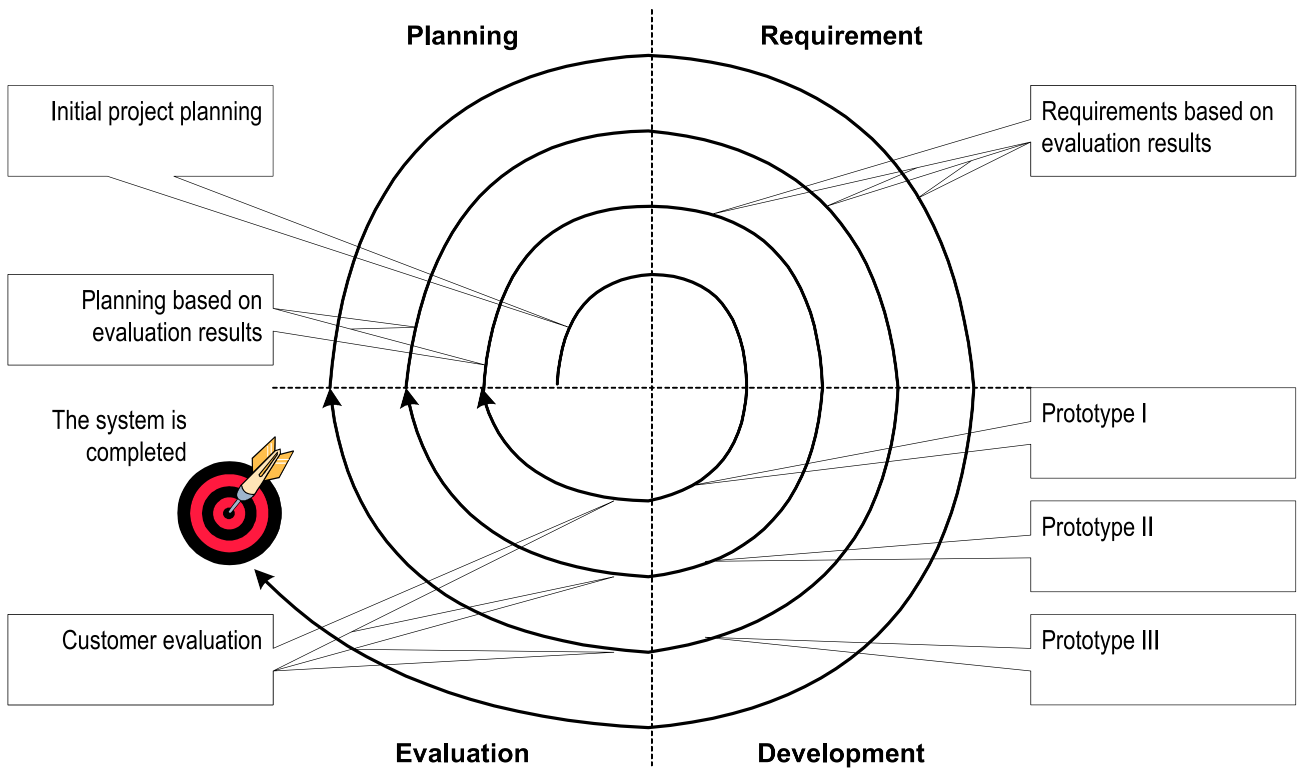

Fig. 1 Spiral process model

To facilitate the spiral model, we adopted the concept of "use case driven." With this concept, the development process was divided into several iterations driven by use cases. In each iteration, only partial use cases are developed. As proceeding more and more iterations, the system will be more and more close to the completion. There is another advantage in adopting "use case driven." A use case and its test case are coupled. When a use case is developed, the system implementing it is treated as a black box and the corresponding test case is developed. Simultaneously developing use case and its test case makes the testing procedure more independent and powerful.

By using class diagram, we depict the software architecture in object-oriented way. In such representation, the interface and functionalities of each class are clearly defined. Further, the static relationship between classes is also described.

The BMS software development process is summarized in Fig. 2. In Fig. 2, it can be found that there are two paths from the beginning of the development: a designing path and a testing path. For the designing path, there are the requirement stage, the module design stage, the detail design stage, and the implementation stage. For the testing path, the test plan, test design, and the test implementation stages are required to generate a unit-test plan. After each use case has been designed and tested, the process will move to the debugging and testing stage and the test report will be generated. The purpose and the deliverables of the design and testing procedures will be described in the following subsections.

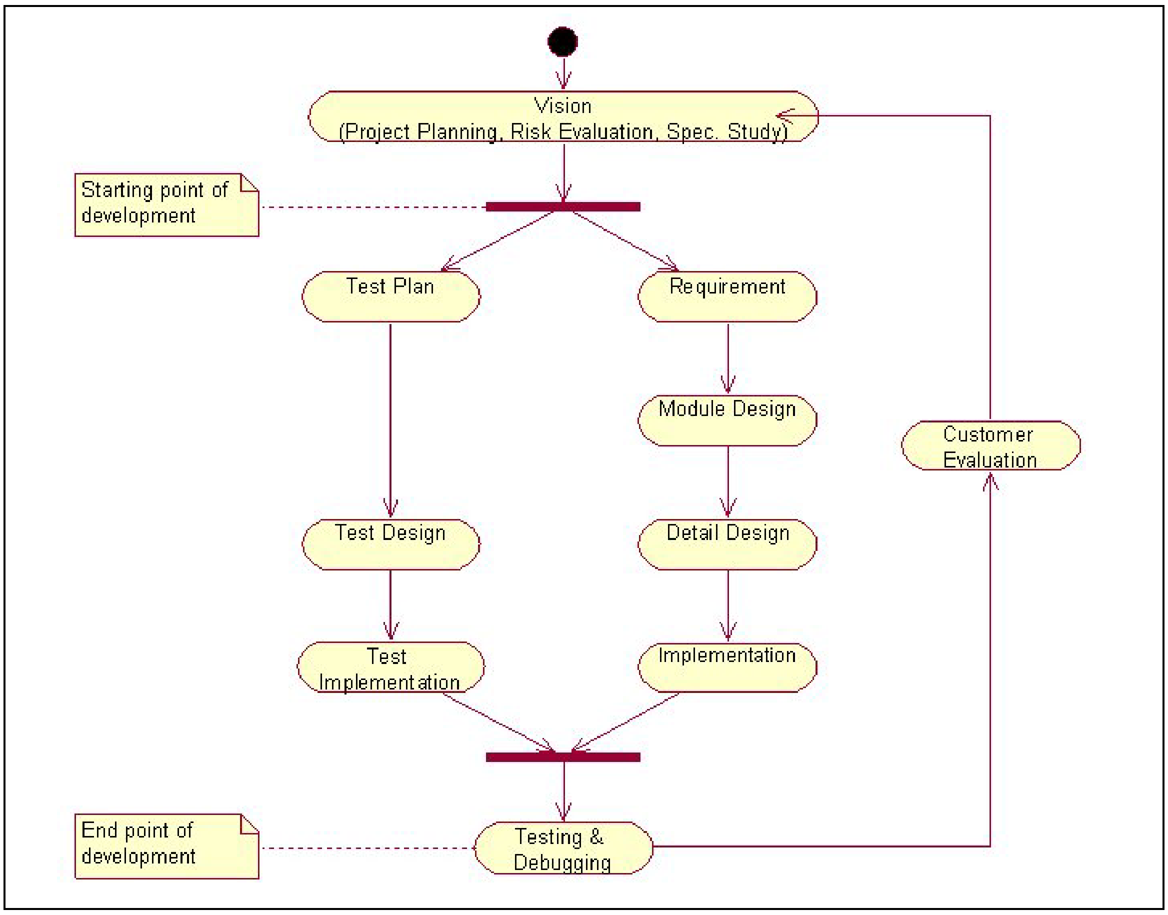

Fig. 2 BMS software development process

## 2.1 Design Procedures

### 2.1.1 Requirement stage

At the requirement stage, we will define the requirement such as the services provided to the outside of a layer and the constraints of the layer. Use case model is used to describe the requirement. The deliverable of this stage are the use case models including use case diagrams and inter-layer message sequence charts (MSC).

### 2.1.2 Module design stage

We start to implement use cases at the module design stage. The purpose of the module design stage is to find the objects that construct the system. The class diagram is used to describe the static relationship between the objects. We also use the MSC to describe the dynamic relationship for each scenario. The attribute and the operation of each object will be found at this stage.

### 2.1.3 Detail design stage

We will specify the detail behavior of each object. The behavior of an object can be described in terms of an extended finite state machine (EFSM). The specification and description language (SDL) is used in this stage for detail designing. Each active class obtained at the module design stage is then mapping to a process in SDL and we will design and simulate the state machine of each process in details. Most of the bugs at the designing phase could be found with the aid of SDL simulation.

### 2.1.4 Implementation stage

The job at the implementation stage is to writing the C codes based on the deliverables obtained at the module and detail design stages. The code of each module should pass the module test by the programmer himself before filing for integration.

## 2.2 Test procedures

As mentioned before, each use case has its corresponding test case. When the use case development passed through the design path, the test case went through “Test plan,” “Test design,” and “Test implementation” stages in parallel.

### 2.2.1 Test plan stage

At the test plan stage, we define the test case for the corresponding use case. According to the inter-layer scenarios depicted in terms of MSCs, the test plan is documented. For each scenario, the purpose, initial conditions, expected sequences, input data, expected results are defined.

### 2.2.2 Test design stage

After test plan is ready, we translate it into scenario file for performing SDL simulation. The simulation will produce SDL-test report, including simulation environment description, scenarios, generated MSCs, related simulation files and a list of test results. Usually, the deliverables in previous stages, such as class diagrams, SDL design and test plan need modification for bug fix.

### 2.2.3 Test implementation stage

The program for testing C code is generated base on the test plan. Similar to previous stage, the testing will produce C-test report.

### 2.2.4 Testing and debugging stage

After individual use case is tested, all use cases should be integrated together and then tested. It is the end of an iteration of development process.

# 3 Guidelines for Implementing the Development Process

## 3.1 Requirement Stage:

### 3.1.1 Layer Planning Guidelines

At the requirement stage, we may perform the layer planning. The input for the layer planning is either a mission of layer development or feedbacks from the previous iteration. The artifacts to be generated are the layer planning document and the UML use case diagrams. During this phase, the engineer has to study the related specifications and to find the required use cases.

During the studying of the related specifications, the engineer has to list the corresponding specifications he has to study before starting his job. For example, 3GPP specifications of 25.301, 25.303, 25.304, 25.331, and 25.921, etc. are directly related to the UE RRC layer. In some cases, more than one engineer will work together to implement a single layer. The engineers may have to indicate the related sections corresponding to his job.

After studying the related specifications, the engineer has to analyze his task and tries to partition it into several use cases. The use cases are then divided into groups based on the priority of the implementation. The use case belonging to the first priority group shall be implemented at the first iteration. Therefore, these use cases should be defined as clearly as possible. For the use cases belonging to lower priority groups, their definition can be refined at the following iterations. In this phase, the only thing we have to do is to find the use cases and draw the use case diagrams. The details of each use case will be determined at the use case planning and modeling phase.

The manpower and the schedule required to implement each use case is then decided according to its complexity. The checkpoints should be decided while planning the schedule. Examples for checkpoints are SAP documenting, detail design & Simulation, coding & unit testing, integration. The time required for the reviewing process should be included in the schedule. One should notice that the designer and test planner should be assigned for each use case. A rule of thumb is that the manpower for designers is roughly twice than that of test planners.

### 3.1.2 SAP Documenting Guidelines

The engineer may have a whole picture about his task after writing the layer planning documents and reading the related specifications. The next step is to define the Service Access Points (SAP) to provide common interfaces to colleagues who are involved in the same project. He has to find the interfaces between use cases and between protocol layers. The interfaces are then documented in terms of the SAP format. The SAP document should consist of, at least, the primitives and data types used to communicate between layers.

The engineer has to understand the details of specifications before drafting the SAP documents. The scope of SAPs can be confined to the on-going use cases that would be implemented at this iteration. Generally, there is an SAP document for each layer. Engineers for the on-going use cases within a layer are responsible for drafting the SAP document and leaders for adjacent layers are responsible for reviewing it.

### 3.1.3 Use Cases Planning and Modeling Guidelines

The last phase of the requirement stage is related to use case planning and modeling. The input of this phase is the assigned use case in layer planning and its related SAPs. The engineer has to plan this use case, describe the requirements of this use case, and depict the inter-layer scenarios in terms of MSC. The artifacts of this phase are the use case planning and the use case model.

In the planning of each use case, we need to list the technical reference that the use case refers to. The section numbers as well as the version number of the referred specification should be provided. The engineer will estimate the time required for implementing this use case, including the use case modeling, module design, detail design and simulation. The time required for the reviewing, including inspection and walkthrough, should be taken into account. The constraint or supporting required for implementing this test case should be pointed out at this phase. The project leader must assign the correct version for the documents describing the working guidelines, templates, standards, and checklists that the project will refer to. Any exception for using of the referred documents should be identified and be approved by the project leader.

In the modeling of the use case, the first step is to find actors. The actors are the layers that interact with the target layer for the test case. For a use case of RLC, the RRC and MAC layers are the actors. Then we have to specify the requirement of the use case. The requirement includes the section number of the specification that the use case is referred to, the supported and non-supported functions, capacity constraint, performance constraint, timing constraint, and the other constraints. The final step is to depict the inter-layer MSC. All of the possible scenarios, including normal and abnormal scenarios, for this use case should be found at this phase. The constraint for each scenario should also be indicated. Generally, each use case will have its corresponding inter-layer MSC.

## 3.2 Module Design Stage

### 3.2.1 Module Design Guidelines

The input for the module design stage is the use case model. The artifacts are the class diagram and the inter-class MSC. The engineer has to define the participating classes, depict inter-class scenarios in terms of MSC, and define details for each class.

To define the participating classes, we have to find the classes within this use case. There are three types of classes: boundary class, control class, and entity class. The boundary class is responsible to the interaction with the actors. Normally, there would be at least one boundary class for each actor. There may be some boundary classes that are shared by several use cases. For each layer, the boundary classes are used to increase the protocol portability. It can be used to separate the core function of the protocol from the framework and the RTOS related functions. The control class is responsible for controlling the overall flow of the use case. Normally, each use case may have its dedicated control class. We should prevent form sharing one control class among several use classes. The entity class is a long-lived database that used to storage permanent information. Generally, they are tables that shared by different use cases.

Then we have to find the shared classes among use cases. For example, the data stored in an entity class may be referred by several use cases. These classes are named reusable classes and we may use either one of the two methods proposed below to design them.

The first method is generalization. We can first find classes for each use case individually. After the classes have been found, we can try to combine several classes with similar functionality into a single class. It's a little bit time-consuming but is suitable for less experienced engineers. The second method is try to adopt the inheritance concept. A single class is defined based on the common need of several use cases. It may not have to spend effort to define several classes with overlapped functionality but only suitable for experienced engineers.

In order to describe the interaction between classes or objects, the inter-class scenarios are required. One should notice that the description of the inter-class MSC should be based on the requirement defined by the use case model. The arguments and types of each operation should be included while defining the message.

The details of each class should include operations, attributes, constraints, and specific relationships. Some of them may not be completed at the module design stage but have to be finalized before the end of the detail design and simulation stage. The public operations of each class can be found during depicting the MSC. The engineer has to specify the way to meet the constraints for capability, performance, and timing for each use case. Last of all, we may need to use extra class diagram and/or aggregation relationship to describe a complex data structure.

## 3.5 Detail Design & Simulation Stage

### 3.5.1 Detail Design & Simulation Guideline

Now we have the class diagram, inter-class MSC, and unit-test plan at hand and have a full understanding for the specification to be implemented. At this stage, we need to produce the SDL design, perform some basic simulations and record the results. The SDL and the test reports will be inspected according to the class diagram and the inter-class MSC obtained at the use case planning and modeling phase. Based on the simulation results, some errors could be found and the class diagrams should be refined accordingly. The details procedures required at this stage is described below.

The steps to construct a skeleton of the SDL design consist of: include the common definitions that used by the other layer, map the system architecture defined in the UML design to SDL design, and complete the intra-layer definitions. We have designed a tool that can convert the data types defined in SAP to an SDL pr file. The converted file should be named as 'XXXYYY_Service_Primitives.pr,' where XXX and YYY represent the two layers that interact with each other. One of them must be our targeted layer. Beside of this, there would be a common definition of the data types that used by many SAPs. The global definitions are converted in a similar way to produce another SDL pr file, named 'General.pr.' 'XXXYYY_Service_Primitives.pr' and 'General.pr' are the two common definition files that should be included. SDL is used here to describe the detail operation of a use case. We have to map the software architecture in the UML design to the SDL design. Table 1 defines the mapping from the UML units to SDL units.

| **UML** | **SDL** |
|---|---|
| Boundary class | Process in Block 'Boundary' |
| Control class | Process in Block 'Use case' |
| Entity class | Procedure partition in Process 'XXX_Global' of Block 'Exported_Process' |
| Other class | Same as Entity class |
| (Public) operation | Procedure in the partition for public operations |
| (Private) attribute | Local variable in the local variables partition or in the variable declaration partition |
| (Private) operation | Procedure in the partition for private operations |

Table 1 Mapping between UML and SDL units

- Classes → Processes or Procedure Partitions: In UML design, the most often used classes are boundary class, control class and entity class. We put all processes for boundary classes to the block named 'Boundary.' For the use case that has been divided into several packages, each package is mapped to a block and its associated control class is mapped to a process within the block. Otherwise, each use case would be mapped to a block and its control class is mapped to a process within the block. For the case that several use cases share a common entity class, therefore, we need to have a block named 'Exported_Process' to keep the share information. All of the entity classes are mapped to the partitions (including 'Variable declaration partition,' 'Public operations partition,' and 'Private operations partition') within the process named 'XXX_Global' such that the 'Public operations partition' can be shared using SDL import function. For the remaining classes, the mapping rule is the same as that of entity class.

- Operation → Procedure: each operation in UML is mapped to a procedure in SDL. Several operations can be grouped to a procedure partition. For the control class, its operations are kept in the procedure partition of its corresponding process. For the entity class, its operations are kept in its corresponding partition. As for the arguments of the operation, they are mapped to the parameters (i.e. IN or IN/OUT) of the procedure.
- Attributes → Local Variable Partition: The attributes in UML is mapped to the local partition. One should note that the attributes of a class must be put in the same partition. The location of the partition will be dependent on the class type they belong to.

The last step for designing the SDL is to fill in the definition for constants, data types, imported procedures, signal and parameters.

Then we can start to describe the finite-state-machine (FSM) behavior of the operation defined in the specification using SDL. The variable declaration and the flow of the procedure should be carefully designed such that they can be easily implemented using C language. Simulation is used to verify our SDL design and to reduce the bugs at the early developing stage. All of the test cases defined in the unit-test plan have to be simulated. For each test case, we should have the corresponding setting for the simulation environment and a scenario file for regression testing. The scenario file is used to trigger the SDL state machine to the initial condition requested by the test case.

During the simulation, bugs would be found and fixed. The details of attributes and operations will also be explored at this phase. Therefore, we need to refine the artifacts produced at the module design stage for consistency. For the UML class diagram, we need to refine the arguments and their types, the return types of private operation, coding consideration, and comments for public and private operations. The coding consideration should indicate the C language implement issue. For example, call by value or call by pointer will be used. The comments will focus on the functional description for each operation. As for the attributes, we need to specify the type definition, comments and the coding considerations. For the inter-class MSC, we should use the public operation to describe the message on the MSC. It could be better if the arguments of the operation can be addressed.

Some of the table operations may be easy, some of them may not be easily described in terms of SDL. Therefore, we may not be able to use formal SDL representation to implement them. Informal SDL with additional documentation would be chosen as the detail design for these table operations.

However, we should keep this part as clear as possible to prevent from misunderstanding. By the way, figures can be used to assist the description for complex operations and data structures.

### 3.6 Implementation Stage

### 3.6.1 Coding & Unit-testing Guidelines

After we fix the bugs found in the detail design stage, we can implement our design (i.e. the class diagram and SDL) using C code. The framework of .c and .h files are obtained from class diagram, as shown in Table 2. Each class is associated with a .c and a .c file. The EXTERNAL GLOBAL FUNCTION of the .h file is used to declare the class of public operation. The static variables, static functions of .c file define the private attributes and the private operations of the class, respectively. The global and static functions define the prologue, return type, name, argument list, and braces of a function. The include list of .c file is obtained based on the association between classes.

| **UML** | **C** |
|---|---|
| Class | One **.c** + One **.h** |
| (public) Operation | **extern global function** of .c |
| (private) Attribute | **static variable** within .c |
| (private) Operation | **static function** within .c |
| Association between classes | **#include "xxx.h"** |

Table 2 Mapping from UML to C language

## 4 Summary

In this paper, we described the protocol development process used in BMS step by step. The process incorporates the concepts of "spiral model," "use case driven," and "object-oriented analysis and design" and adopts three kinds of graphical languages. With the process, the precision and traceability of projects can be ensured without the burden of too much documentation.

## 5 Reference

[1] Ivar Jacobson, Grady Booch, and James Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.

[2] Roger S. Pressman, *Software Engineering: A Practitioner's Approach 5th ed.* , McGraw Hill, 2001.

[3] Terry Quatrani, *Visual Modeling with Rational Rose 2000 and UML*, Addison-Wesley, 2000.

[4] Bruce P. Douglass, *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*, Addison-Wesley, 1999.

[5] Jan Ellsberger, Dieter Hogrefe, and Amardeo Sarma, *SDL: Formal Object-oriented Language for Communication Systems*, Prentice Hall, 1997.

[6] VERILOG, *ObjectGEODE Method Guide v1.0*, VERILOG SA, 1996.

[7] VERILOG, *ObjectGEODE SDL & MSC Editor User's Guide v4.1*, VERILOG SA, 1999.