ATM software analysis and design using OMT

Abstracts
This article describes how to apply Object Modeling Technique (OMT) to analysis, design and implement a simple ATM switch using C++ programming language. Practical design considerations of applying OMT methodology in software development are discussed.

Introduction
Object oriented analysis and design (OOA&D) was a hotly debated subject in the early 90s. With the maturity of the methodology and its supporting tools, OOA&D has replaced structured analysis and design methodology to become the dominant software development methodology used by various industries for software development.

Among all the available OOA&D methodology, Object Modeling Technique (OMT) developed by James Rumbaugh and associates [Rumbaugh 1991] is one of the most widely accepted OOA&D methodology with more than 20 vendors producing various kinds of OMT development tools (e.g., Rational, Objecteam etc...). OMT is especially useful for large software project development due to its problem abstraction capability. There are a lot of successful software projects developed using OMT methodology [Cheung 1994].

This article describes how to apply OMT to analysis and develop software for a simple Asynchronous transfer mode (ATM) switch [McDysan 1995]. Since the objectives of this article is to show how to apply OMT in system analysis and design, important ATM switch design considerations and details such as resource protection and abstraction and service and network interworking with other protocols etc... are ignored in the model.

Requirements
The hypothetical and simple ATM switch to be developed in this article has 12 ATM ports. Ten of them are used to connect customer premises equipment (CPE) and the remaining two are used for network trunk connections to connect the switch to the ATM network. Each port or connection supports 1024 virtual circuits (VC). Signalling messages from a CPE is sent at VC 5. The switch supports only one-way ATM setup and release requests from CPEs. Figure 1 shows a typical network configuration using the simple ATM switch.
Figure 2 shows the setup and release protocol supported by the simple ATM switch. It is the only protocol sequence supported by the switch.

**Figure 1 Network configuration using the simple ATM switch**

**Figure 2. The only protocol supported by the simple ATM switch**

*connection setup and release between CPE-B and CPE-M*
In Figure 2, when CPE-M accepts the connection setup request by responding a connect message to CPE-B, the simple ATM switch will setup its ATM fabric for the session and select a free VC (i.e., bearer_vc_assigned) for bearer traffic exchange between CPE-B and CPE-M.

We now have the product requirements, let roll-up our sleeves and apply OMT to analysis, design and implement the simple ATM switch.

In OMT, there are four distinct development phases with different deliverable. Table 1 summarizes each of the stages and its deliverable.

### Table 1: Software development phases and deliverable using OMT

<table>
<thead>
<tr>
<th>Phase</th>
<th>Deliverable</th>
</tr>
</thead>
<tbody>
<tr>
<td>System analysis</td>
<td>Object, dynamic and functional models</td>
</tr>
<tr>
<td>System design</td>
<td>refined models that meet the desired technical, business and architectural constraints</td>
</tr>
<tr>
<td>Object design</td>
<td>refined object, dynamic and functional models with implementation details</td>
</tr>
<tr>
<td>Implementation</td>
<td>executable programs</td>
</tr>
</tbody>
</table>

OMT emphasizes a continuous development process. Each stage refines and adds more details to the models developed in the system analysis stage until the models become meaningful for coding. In summary, as with any object-oriented methodology, OMT is primarily a process of iterative refinement and adding details.

### System analysis

The goal of system analysis in OMT is to understand the problems or requirements and to develop models that represent what system will do. Three models are developed during the system analysis stage namely the object, dynamic and functional models to represent the objects and their relationship, dynamic control flow and functional transformations of the system respectively.

#### Object model

This is the most important model in OMT. It shows the static data structure of the real-world system and organizes it into workable pieces. It describes real-world object classes and their relationships to each other. Figure 3 shows the object model of the simple ATM switch.
The transformation from the requirements to the object model requires the designers to have a certain level of knowledge in the problem domain (e.g. in this case, ATM signalling and switching etc..) though some methods can help such as to identify noun from the requirements and refine the set until they make sense in the object model. No matter which method is used to develop the object model, none can replace designers’ knowledge and experience in that area.

The above object model reads as follows. The session manager class consults with the bandwidth and route manager classes. It manages many session classes and it interacts with 12 port classes. The bandwidth manager class controls one ATM fabric class. The ATM fabric class has 12 port classes. User and network port classes are a kind of Port class (inheritance). The Port class has 1024 virtual circuit classes and they are controlled by the ATM fabric class. Note that Port class is an abstract class. It has no direct instances but whose descendent classes (e.g., user and network port classes) have direct instance.

**Dynamic model**

The dynamic model shows the time-dependent behavior of the system and the objects in it. Normally, message sequence charts are drawn before producing dynamic models of each class defined in the object model. Note that exceptional events are not shown in the message sequence charts for brevity but are detailed in the dynamic models. Figure 4 shown the message sequence chart of the simple ATM switch to support the setup and release protocol.
Figure 5 shows the dynamic model of the session manager. Note that each class defined in the object model should have its dynamic model. The dynamic model of the session manager is self-explanatory.

Legend:
- internal messages within the simple ATM switch
- external network messages to and from the simple ATM switch

Figure 4. Message sequence chart of the simple ATM switch
Figure 5  Dynamic model of the session manager

Legends
- = termination
- = state
error = time-out no response or unexpected msg received
B = originating ATM address
M = destination ATM address
BW = bandwidth desired
**Functional model**
Functional model is normally constructed after the object and dynamic models. The functional model shows how values are computed, without regard for sequencing, decisions or object structure. Figure 6 shows the functional model of the simple ATM switch.

There are three processes in the simple ATM switch namely the parse signalling message, find destination and allocate resource that transform input and output values. Input and output values are parameters of events between the system and the outside world.

**System design**
The system analysis phase determines what the implementation must do and they are represented by the object, dynamic and functional models. The system design phase determines the high-level strategy for solving the problem and building a solution. For large software project, subsystems are identified and partitioned in this stage to make the development more manageable.

All technical, business and architectural constraints or requirements presented in the requirements and/or business plan are examined in details in the system design stage. The high-level solution selected in the system design should facilitate the subsequent development to realize the product
differentiator identified in the requirements.

The deliverable of the system design is the three refined analysis models that satisfy all the technical, business and architectural constraints described in the requirements. High-level design decision such as the use of COBRA or distribute signalling processing should be factored into the analysis models. It should be noted that the system analysis and design phases are highly iterative. It is not uncommon to oscillate between these two stages for 10 to 20 times before the final models are hammered out.

In theory, the models developed in the system analysis stage should be well-decoupled from the actual design no matter what the high-level solution or architecture is chosen in the system design stage. It has been the author’s experience that this only works for small software projects. For large software project development, some high-level design decision such as the use of distributed signalling processing will no doubt affect the analysis models and the models should be revised to reflect to the actual system that is going to develop.

Object design

The classes or objects discovered during the system analysis and design phases are very high-level without much details. The object design phase determines the full definitions of the classes and associations used in the implementation, as well as the interfaces and algorithms of the methods used to implement operations. It is the stage that designers change their design perspective from application to computer domains.

New objects, objects’ attributes and operations are usually found and added in this stage to facilitate the subsequent implementation. Software designers must choose among different ways to implement the objects to minimize execution time and memory required.

Figure 7 shows the refined object model of the simple ATM switch with attributes and operations defined. All attributes and operations of the classes can be derived directly from the object, dynamic and functional models obtained in the system analysis and design stages. For example, the route manager class has a member function called find_destination() as the message sequence chart and the dynamic model of the session manager shown in Figures 4 and 5 respectively reveals this requirement in the route manager class.
Implementation

Continuous from the object design stage, we can now map the models into executable programs using any object-oriented programming languages. The following illustrates the mapping using C++ programming language.

The operations and attributes defined in the refined object model shown in Figure 7 are simply member functions and private data of the class in C++. Dynamic models defined for each class constitute the program control flow of the classes. Program control flow can be easily imple-
mented using C++ programming features such as if-else, switch, exception handling etc..

Most object-oriented programming languages do not support association relationships defined in the object model. There are several ways to circumvent this programming languages’ deficit in C++ such as using pointers or templates [Linenbach 1996]. Figure 8 shows the session and session manager class definitions with the manage association relationship implemented using pointers.

```cpp
const int MAX_SESSION = 2000;           // max sessions supported by the ATM switch
enum SessionState {IDLE, SETUP, CONNECT, CONNECT_ACK, RELEASE, RELEASE_COMPLETE};
enum Status {INACTIVE, ACTIVE};

class session{
    private:
        SessionState state;
        Status status;
        class session_mgr *sm_pt;
        friend class session_mgr;
    public:
        session();
        ~session() { status = INACTIVE; }
        void enable_session() { status = ACTIVE; }
        void disable_session() { status = INACTIVE; }
        class session_mgr *get_SessionManager() { return sm_pt; }
    };

class session_mgr {
    private:
        session *sess_pt[MAX_SESSION];
    public:
        session_mgr();
        ~session_mgr();
        int setup(char *originating_address, char *destination_address, int bandwidth_desired);
        int connect(int vc_assigned);
        int connect_ack();
        int release();
        int release_complete();
        void add_session(session *);
        void delete_session(session *);
        session *get_session();
    }

    Figure 8. C++ class definition of the session and session_manager classes with attributes, operations and association relationships defined
```

The above pointer implementation of OMT’s association relationships is rather primitive. Fortunately, most OMT design and analysis tools such as the Rational’s ROSE C++ OMT package generate C++ template code for association relationships. This relieves the burden of software designers as association relationships should indeed be a programming language’s feature.

Conclusion

This article uses the design of a hypothetical and simple ATM switch to summarizes OMT software development methodology and processes. A real-world ATM switch is of course much more complex than the one shown in here. For example, a contemporary ATM switch runs PNNI routing protocol for ATM traffic routing and the PNNI software comprises about 100,000 lines of code.

OMT encourages and enforces upfront software analysis and design. In order to produce the three
models defined in the system analysis and design stages, designers have to have a good understanding of the problems and/or the products to be developed. This prevents designers from coding premature. Also, the use of the widely accepted OMT notations and methodology improves communication among software designers. This is especially important for large software development project, which may involve hundreds of software designers across multiple sites. All these transform into a higher quality software that require less time to develop and maintain. Software or objects reuse can also be another big benefit of using OMT. However, software reuse requires a lot of persistent and dedication from management and designers. It is more a management than a technical issues.

Reference


Author biography

Derek Cheung works as the software architect in the Advanced ATM system development division of Nortel Technology (Bell Northern Research) in Ottawa, Canada. He has designed and managed various large-scale network and software development projects. He can be contacted at dereke@nortel.ca.