Building MIB Applications

Ola Ödling

Stefan Wallin

Erisoft AB, Sweden

email: Stefan.Wallin@sa.erisoft.se

tel + 46 910 51700

fax + 46 910 35817

NOMS '94

Background

- ❑ **Building Telecommunications Systems According to the TMN/OSI Systems Management Standards.**
- ❑ **Large projects**
- ❑ **Customer specified Information Models**

NOMS '94

This presentation will focus on the technical issues of building a Network Element Agent according to the TMN and OSI Systems Management Standards. Particularly aspects of implementing the MIB are covered. A number of tools produced by Erisoft to support that process are presented.

Erisoft is an Ericsson company with about 300 employees, specialized in software for telecommunications systems. We have around 100 people involved in projects designing and implementing MIB Agents. The products are various SDH and PDH equipments.

Characteristics of the development is that each new customer has required their own Information Model to control the Network Element, furthermore the draft standards are still changing so there has been the need of continuous changing and replacing the Information Models in the systems. This need forced us to search for tools that would help in the process of developing the MIB software. Since there were no satisfying existing products we ended up in producing tools on our own. These have proved to significantly reduce the effort of producing the software.

Contents

- ❑ **IM Specification**
- ❑ **GDMO Tools**
- ❑ **Software architecture**

NOMS '94

- ***IM Specification:*** issues on Information Model specification and GDMO/ASN.1.
- ***GDMO Tools:*** description of the Erisoft MIB/GDMO tools.
- ***Software architecture:*** issues on software architecture for an Agent.

IM Specification

- ❑ **Interface specification or functionality specification?**
- ❑ **Readability & Understandability**
- ❑ **Informal behaviour specification**
- ❑ **Consistency**
- ❑ **Stability**

NOMS '94

The use of an Information Model¹ as a Requirement Specification is in some communities unclear. An IM could be seen as a functional requirement specification with high detail on syntax and low detail on semantics. There seems to be two alternative ways to resolving this problem. One is to map an IM model to a separate interface independent functional specification. Another alternative is to increase the semantics within the IM to contain more details on the semantics of operations.

When Developing an Agent Application the central specification for that software is naturally the IM specification. Critical for the success of the project then is that the specification is accessible and understandable for the project members. This is however a problem: the size of an IM, about 500 - 1000 pages, makes it hard to distribute and update. GDMO and ASN.1 is not easy to read and understand. In order To find one piece of information you have to follow a number of references simultaneously.

Though an IM may look very formal it is not, since the important behaviour specification is informal. This problem was covered by a number of papers at the Integrated Network Management conference².

Another problem is that the IM specifications are seldom verified syntactically by a GDMO and ASN.1 compiler.

The current status of the different Information Models are not stable. Every telecommunication administrator with self respect wants their own IM.

1. ISO 7498-4, ISO 10040, ISO 9595, ISO 10164-X, ISO 10165-X

2. Hegering, Yemeni, Proceedings of the IFIP TC6/WG6.6 Third International Symposium on Integrated Network Management. San Francisco, CA USA, 18-23 April, 1993.

IM Specification formalizing behaviour

❑ Example:

Action foo:

Precondition:

administrativeState == unlocked

Postcondition:

usageState == busy

Response:

ActionResponse (parameters)

AttributeValueChange (attribute)

NOMS '94

All our projects begin by formalizing the behaviour specification by using assertions¹ and stimuli/response models.

In order to formalize the specification assertions are connected to each management operation. These are in terms of pre- and postconditions on attribute values of the managed object.

As the consequence of a management operation not only the operation response is important, also all side-effects has to be specified, e.g. attribute value change notifications, object creations/deletions etc. These are specified by a stimuli/response model.

1. Bertrand Meyer Object Oriented Software Construction Prentice Hall, 1988.

GDMO Tools

- ❑ **Code generator, MOC**
- ❑ **Interactive browser, MOB**
- ❑ **Test tool, MOT**
- ❑ **Editor, MOE**

NOMS '94

The issues previous presented resulted in the development of a number of tools. We had to develop them ourselves since no satisfactory solution could be found on the market.

The central tool to develop was a code generator that from an Information model could generate the kernel of the Agent. We choosed to use the Retix ASN.1 compiler for the ASN.1 part and developed our own GDMO compiler. This compiler does also parse and generate code for the ASN.1 part since we want the generated types to be efficient to use inside the application and not optimized for a protocol which is the case with an ASN.1 compiler.

The problem of accessability and understandability was solved by developing an interactive IM browser. Soon the need for editing support for Information Models become critical so we integrated emacs and Framemaker to the browser. from the browser you can directly reach the correct part in the document and with in the editor you can perform verification of the syntax and static semantics of the Information Model.

In order to support the huge amount of testing needed to verify an IM interface we made a tool that automatically generates test drivers and evaluators from a test specification.

MOC - MO Compiler

NOMS '94

This tool is dedicated to off-line create a number of files, being the kernel of an Agent or Manager in a TMN System. The main purpose is to avoid to manually rewrite the Agent or Manager every time a new IM is to be realized, or an old one being revised.

The compiler uses one GDMO/ASN.1 coded file, describing the information model, an Application Specific File (ASF) for clarifying the wanted configuration e.g. which optional packages to use, template files being a blue print for the output and user-defined files implementing the types that cannot be generated automatically.

There is no knowledge in the code generator about the target software architecture, instead the code generation is described in the 'Template Files'. The MOC uses the template files as blue prints, i.e. copies its content to the corresponding output file, except where keywords are present. Keywords are entries into the symbol table of the information model. In this way the design of the manager/agent is expressed as a number of template classes without reference to an explicit IM. Wherever such a reference should be needed a keyword is inserted instead e.g. <Attribute>, results in that the generated code will contain a part for each attribute in the specific IM.

The connection between the IM dependant part and functionality in the application is also specified by the ASF. In this file a connection between a MO operation and an application function is bound. E.g. if you want an action to call a function in your application the action is connected to a C++ funtion in this file. This can be compared with call-backs used in GUI tools. By having this in a separate file we keep the application dependency out of the IM and the IM dependency out of the application.

MOB - Managed Object Browser

- ❑ **Interactive tool to navigate an Information Model**
- ❑ **Verifies the GDMO and ASN.1 files**
- ❑ **Graphic presentation of Inheritance and Name Binding**
- ❑ **Generate 'User Friendly' IM**

NOMS '94

An information model for a typical system is a binder of about 500 pages. The amount of information is huge and very detailed. There is often just a few people in the project that knows how to find the information in such a binder. These people become critical in the project, and furthermore it often takes a large amount of time to find the information, even for experts, since the specifications use references and inheritance in a number of levels. "Ten fingers is never enough".

Therefore we made a tool that lets you browse the information model directly. Finding information that could previously have taken hours are now solved in seconds. Furthermore introducing new people to an information model is much easier when using an interactive browser.

Complete graphs of the inheritance structure and name binding structure is often missing in the specifications so the tool automatically generates these graphs.

Another need was to be able to produce 'User Friendly' IM documentation for certain parts of the information model. The project are often divided into teams e.g. one team working with Fault Management, FM. Then the tool can generate a document containing only that part of the IM. This document is not in GDMO format, instead in a more straight forward easy to read format, where the inheritance structure are flattened and the levels of indirections are minimized. This document is in FrameMaker format. Adaptions to other documentation tools is possible.

MOT - Managed Object Tester

- ❑ **Supports test specification and automatic test sequences to be executed via a Q3 interface based on CMISE/CMIP protocol.**
- ❑ **Test drivers and evaluators, (C++ code), are generated from the test specification.**

NOMS '94

Imagine an information model of 500 pages, then imagine to write test programs for that information model. One soon realizes this is a dead end if serious testing is to take place. So for that purpose we have developed a MO test environment, which automates as much as possible. One problem with a test specification with a IM as input is that an information model is too informal in order to fully generate the expected output. E.g. an action does not specify which attribute value changes it causes. So for that purpose we first of all formalize the IM adding formal behaviour.

With that as a base test specifications are written in a format similar to CMISE primitives. The test specifications are then fed into MOT which generates test drivers and test evaluators. The test driver issues requests and monitors event reports and responses over the Q interface to which an agent is connected. When running the evaluators test records and test logs are updated/created.

Fundamental to a successful test environment is also to have connection to the CM system so that at any point you now how much of what system version is tested against what test specification.

Software Architecture

- **Separate IM software clearly from application software**
- **Risk for data redundancy**
- **Facilitate easy change of IM dependant software**
- **Need for OO database supporting large amounts of data**

NOMS '94

The most fundamental issue about an architecture for an agent application is to keep the pure application functionality separated from the information model software. Though it sounds natural when it comes to code it is easy to get these parts dependant. The approach we have taken is to develop one part of the application containing the functionality of the network element fulfilling functional requirements as a separate system. It exports its services via an Application Programmers Interface, API, expressed in C++ header files. There is no knowledge of information models here. Another part of the software has the knowledge of the Information Model and makes calls to the application. This is a traditional approach in application development when separating the user interface from the application. The problem however with this is that there may come to situations where you have data redundancies, say an attribute is stored in the MIB, but also has a representation in the application, if the two values are not consistent, which one is the correct?

The IM part has to be extremely easy to change since still the IM models are unstable and each customer has its own requirements on Information Models.

A Network Element will contain large amounts of data to support a MIB, in our applications up to giga bytes. The need for a database becomes urgent. Prototypes and calculations we have made point in one direction. An Object Database is needed both for performance reasons but also due to the conflict between the relationship model and an object model.

The Future

NOMS '94

The future direction of the IM/GDMO environment will make use of a repository for storing all definitions. Each item, (identifiable by Object Identifier), of an Information Model will then be treated like one unit. This is in order to leave the view where on Information Model is one file. This makes it possible to make configurations out of standards and vendor specific definitions into different systems. It also enables a more incremental editing compiling cycle. This architecture is much like the trends in developmnet environements for modern systems e.g Energize¹.

1. Energize Programming System, Lucid Inc