



Embedded Telecommunications Management Network (TMN) Solutions

Definition

A telecommunications management network (TMN) agent is an application that runs on a network element (NE) that provides one or more management systems the ability to manage the NE. An embedded TMN agent is used to manage a telecommunications device that uses a real-time operating system (RTOS).

Overview

Embedded TMN agents, as defined by TMN standards, are increasingly becoming the standard for network management. But the embedded environment presents many challenges for developers. Besides the challenges of developing for real-time embedded systems, the creation of these agent applications is complex and has many steps. Embedded agent applications use object-oriented technology to perform management functions on real resources on the network.

This tutorial outlines the steps required to create embedded TMN agents. Covering from what an embedded agent is to how it fits into the overall TMN network management structure to the process of creating embedded TMN agents, this tutorial will explain the challenges and methods available to overcome them.

Topics

1. What Is an Embedded TMN Agent?
2. Alternatives to TMN Network Management
3. The Embedded Environment
4. TMN Agents Use Object-Oriented Technology
5. Implementing a TMN Agent
6. Developing the Agent Model
7. Developing the Agent Software

8. Testing the Agent

9. Conclusions

Self-Test

Correct Answers

Glossary

1. What Is an Embedded TMN Agent?

The role of a TMN agent application is to provide one or more management systems the ability to manage an NE. An embedded TMN agent resides on a board that controls a shelf or shelves in an equipment rack that makes up an NE. These boards are often referred to as shelf control units (SCUs) or management processing units (MPUs). The agent runs in an RTOS. This special operating system is small, fast, and inexpensive in large quantities, which makes it ideal for telecommunications equipment. Below is a diagram (see *Figure 1*) of a telecommunications device. The agent running in the SCU performs management functions on the rest of the boards in the rack. The agent receives the management requests from a manager application, which typically resides on a remote system.

Figure 1. RTOS-Based Controller (Agent) or SCU

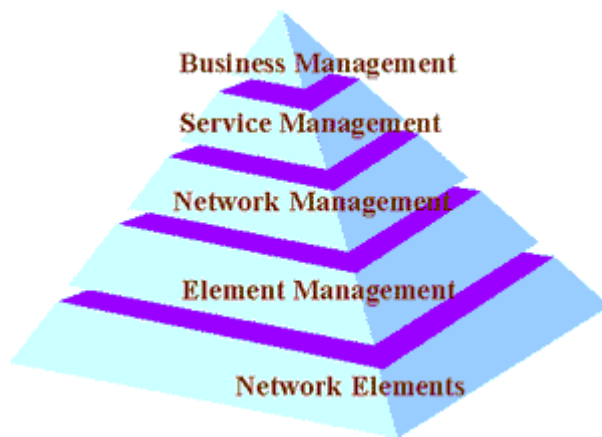


The manager and agent applications communicate using common management information protocol (CMIP) over an open systems interconnection (OSI) protocol stack. OSI, embraced by TMN recommendations, is a standard way for two applications to communicate across the network. CMIP is an object-oriented protocol for management also embraced by TMN recommendations. CMIP over an OSI stack is the TMN standard for communication between manager and agent.

Management systems access telecommunications equipment to perform a variety of management functions on the device. These functions are known in the telecommunications world as operations, administration, maintenance, and provisioning (OAM&P). In the TMN framework, these are broken down into five primary management functions: configuration management, fault management, performance management, accounting management, and security management.

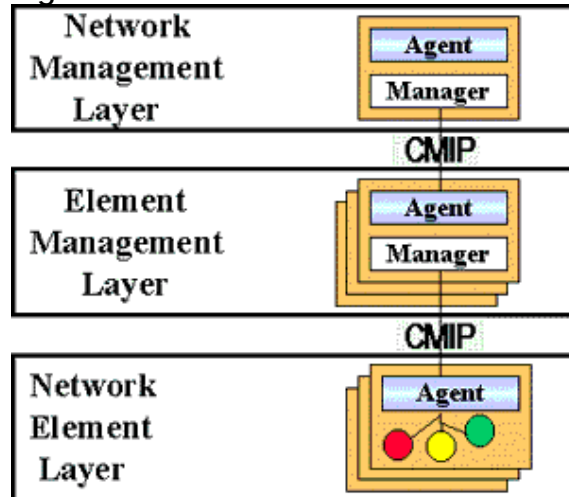
Embedded TMN agents are part of the larger framework of TMN. The TMN management structure is often represented by a five-layer pyramid (see *Figure 2*). Embedded TMN agents are in the lowest layer of the pyramid, the network management layer.

Figure 2. TMN Management Structure



It is important to point out that while management and agent applications exist in management platforms throughout the TMN, embedded TMN agents typically are found only within NEs at the lowest level of the TMN framework (see *Figure 3*).

Figure 3. Lower Levels of the TMN Management Structure



2. Alternatives to TMN Network Management

TMN is a standard way to manage a telecommunications network. But many telecommunications equipment manufacturers still use proprietary or region-specific network management agents, such as transaction language 1 (TL1) or signaling network management protocol (SNMP).

There are various ways to accommodate these agents within a TMN framework. One is to use a mediation device or a Q-adapter. A Q-adapter simply takes a message from a management application and translates it so that an agent application can understand it. In turn, the Q-adapter translates alarms and reports from the agent back to the manager.

Another alternative is to use an adjunct processor for network management. This is almost always a UNIX box that sits between the manager and the device on the network and can manage the NE using a proprietary means while communicating to the manager via TMN. In this case, the adjunct processor hosts the TMN agents. Some equipment manufacturers create adjunct processors that can manage more than one rack of boards. But because the cost of a UNIX box compared to the cost of an RTOS is very high, this type of network management can be very expensive.

There are a few problems with using these methods. For one thing, the mediation device or adjunct processor can be a single point of failure for the network. If that device goes down, the manager cannot communicate with any of the network elements on the network. The use of a system other than TMN for network management also limits the amount and complexity of functions that can be performed. For example, there is no containment or filtering, which actually increases bandwidth used for network management communications.

3. The Embedded Environment

As described in the introduction, an embedded TMN agent is a management application that runs within a RTOS. This environment is commonly referred to as an embedded system and primarily indicates an environment in which applications can be configured to execute in a determined period of time and amount of memory.

An attractive aspect of an RTOS for telecommunications equipment, beyond its performance characteristics, is the cost. This type of system is usually packaged as a development platform along with compilers and debugging tools used for developing the RTOS-based applications. While this development package may be fairly expensive, the cost of license fees for deploying the RTOS itself, along

with any developed applications, is very low when purchased in medium to high volumes.

Embedded applications, such as TMN agents, may be designed and/or configured to use a variety of memory resources. Although embedded environments can include a local disk drive, many do not. It is therefore common for the core application to be stored in read-only memory (ROM) and then loaded into random-access memory (RAM) for execution, because RAM typically provides much faster execution times. Some sort of non-volatile RAM (NVRAM), like flash memory, is used for storage of configuration parameters or variables that must be stored during the operation of the application in case of a re-start.

Because the volume of systems deployed using an embedded environment is usually high, equipment manufacturers strive to minimize the amount of memory of each type that is required for their applications, including the TMN agent. Often these systems will require no more than 32 Mbytes of RAM and may be as low as 8 Mbytes.

These characteristics make embedded systems ideal for use in application-specific telecommunications equipment that is deployed in high volumes and has rigid performance requirements. Although this is appealing to equipment manufacturers, it presents unique challenges for software developers. It is therefore critical that vendors who supply communications software and agent development tools for the embedded environment are well-acquainted with the environment and its associated challenges. It also is important that the tools themselves are designed specifically for the embedded environment.

4. TMN Agents Use Object-Oriented Technology

Object orientation is a technique that allows two objects to communicate with each other without knowing the internal processes of the other object. It also allows programs to operate on objects without knowing the internal operations of the object. This is a good way to do network management because it makes interoperability easy. As long as the operations conform to standards, the object itself can be proprietary, and it can interoperate with another object. Object orientation also allows for salability and reuse, making integration with new technologies easy.

It is necessary to understand the following concepts in order to talk about object-oriented embedded TMN agents:

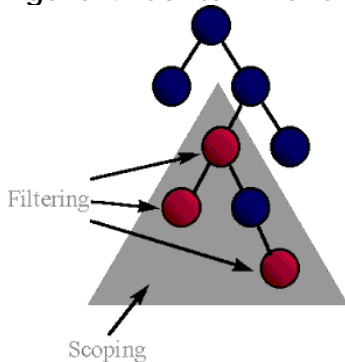
- An object is a software package that contains a collection of related methods (actions that can be performed) and variables (data that can change over time).

- A class is a template that defines the methods and variables to be included in a particular type of object.
- Instances are objects that belong to a class but contain specific values for the variables.

So, in essence, an object is an instance of a particular class, where the method of the object is specified by its class and the value of the object is defined by its instance.

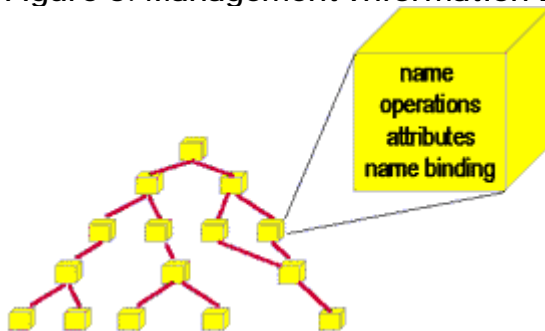
In TMN agent technology, agent development is based on a containment tree (see *Figure 4*), which is a database of named object instances. It uses hierarchical naming, so objects are contained within objects. The containment tree actually holds the object instances that occur on the real resource. This hierarchical control allows for scoping and filtering functionality. Scoping lets multiple objects be operated on at once. Objects within the scope can be filtered based on their attribute values. This makes management much more precise and, as a result, saves bandwidth.

Figure 4. Containment Tree



On the manager side, however, a management information base (MIB) is used (see *Figure 5*). An MIB is an abstract representation of real resources on the network. It provides the manager with an outline of how real resources are going to look and act on each NE.

Figure 5. Management Information Base



The difference between a MIB and a containment tree is important to understand. For example, specific attributes can be defined for a country. It can be said that it has cities, capitals, and states. These terms can be defined specifically. For example, the definition of a city is "an inhabited place of greater size, population, or importance than a town or village." All of this information would be contained in the MIB. Now think of a specific country, such as the United States. With this specific country in mind, one can list specific cities, capitals, and states. This is the containment tree, or the set of real things, as defined in the MIB.

5. Implementing a TMN Agent

There are three basic steps to developing a TMN agent application: developing the information model, creating the actual software code, and testing the agent application.

To develop the information model, it is necessary to use a specification language. The specification language used in TMN to define managed objects is called guidelines for the definition of managed objects (GDMO). GDMO uses abstract syntax notation one (ASN.1) to define data elements in an independent manner for exchange over a communications network.

Both GDMO and ASN.1 are difficult to learn because the syntax is very complicated. But various types of tools exist to help develop the GDMO. Some tools use graphics to help develop relationships between objects in the information model and then automatically generate the GDMO information model so that it isn't necessary to write any GDMO oneself. Tools also may allow for the development of the main structure of the information model using standard managed object (MO) classes, and then one can augment them with particular characteristics, constraints, inheritance relationships, and naming trees that are specific to one's application. Alternatively, one can create the GDMO by hand.

After creating the GDMO model, it is necessary to develop the C++ code for the agent application. Each application that is built must be customized for the specific RTOS on which it will run. Usually this necessitates porting the application to the appropriate RTOS or writing code specifically for the RTOS. It also is necessary to write the code that will associate the GDMO managed objects in the model to the real resources of a device. Finally, it is important to test the agent to ensure that it runs properly on the network. To do this, it is necessary to write a program that will mimic a manager application.

To add to the agent application after it is developed, it is necessary to add to the MIB using GDMO and to repeat all of the steps again. As a result, tool kits that offer iterative development are extremely useful.

6. Developing the Agent Model

The model is the view that the manager will have of the resources on the NE. This view is created in a standard way, using GDMO. ASN.1 is the standard way for the model to see the data.

The first step in creating an embedded TMN agent application is to create a model. It defines the kinds of MOs in the containment tree and how these objects relate to each other. The MOs are arranged in a hierarchy. Each MO has specified attributes and operations. All of this information is defined by the agent model. The agent model will be used in the development of the agent code and for testing the agent.

It is not necessary to create the agent model from scratch. Pieces of standard models can be used or built upon. One could start with the standard model and customize it or add subclasses. Alternatively, one could use a tool to help in the creation of the agent model. Some tools are graphical user interface (GUI)-based, allowing one to graphically represent the relationships between the MOs in the model and then use the tool to generate GDMO automatically.

7. Developing the Agent Software

The agent model is the foundation for the agent application. The code for the agent software is based on this GDMO model. To create the software code for the agent application, it is necessary to create C++ (or whatever language is chosen) classes to somehow represent the GDMO managed object attributes, actions, and notifications in the model. The GDMO that makes up the model references other files called documents. These documents contain the GDMO standards that the model references. These documents also must be incorporated in the code.

This would be a difficult and time-consuming task to do by hand, but fortunately tools are available to automate part of this process. Some tools can build an application based on the GDMO model automatically linking GDMO documents, the model, and other information and creating a header file and a C++ makefile—everything that is needed for the agent application. It is necessary only to tell the tool where all of the pieces are located.

There are agent features that must be included in the software that are difficult to code by hand. The following are features of the agent application:

- The agent application must be able to encode and decode the CMIP protocol data units (PDUs) sent to it. CMIP defines how to do secure electronic transactions (SETs), government emergency telecommunications service (GETS), etc.

- The manager must know the directory structure of the agent. Alternatively, the manager must be able to discover the agent's directory structure, which means that the manager asks the agent what the directory structure is. This is provided in the containment tree.
- The agent application must support allomorphy, an MO class that can look like a different MO class to different managers. To understand this concept more fully, think of a tape recorder/compact disc (CD) player as an MO. If the agent application supports allomorphy, this small stereo can be seen as a tape player to one manager, a CD player to another manager, and both to yet another manager. This way the agent can interact with a wider range of managers. There does not have to be a separate attribute or instance for each manager, which saves bandwidth.
- The application must support single mode fibers (SMFs). SMFs are functions that provide more complex behavior than the normal manager-to-agent commands like GET, SET, etc. Behavior may include event forwarding and attribute modeling. In other words, the agent can perform operations without being told by the manager. For example, the agent can send a signal only when something goes wrong.
- The application must be able to coordinate message processing so that the agent can perform more than one operation at a time or receive more than one message at a time. This interprocess communications (IPC) must be handled because often the resource is not on the same processor as the agent.
- The Q-interface is the interface between the communications protocol and the manager and agent. In the case of embedded TMN agent, the Q-interface is the OSI stack. The agent application must be integrated with this Q-interface.
- The manager must have scoping and filtering capabilities so that it can perform operations or find out information about entire sections of the containment tree. The manager can operate on whole sections of the containment tree. Scoping allows for the identification of a set of managed objects to run the test on, while filtering allows for the display of results within a specified range. This functionality must be built into the agent application.
- The application must be written for or ported to the selected RTOS. Many different RTOSs are available from different vendors, each offering unique functionality. Therefore, the agent must be specific to the target RTOS.

- Finally, it is necessary to integrate MOs with real resources. This can be done in C++ by hand, but some tool kits available can ease that process by providing C++ hooks or stub functions so that all that is needed is to write the C++ code that associates these hooks with the real resources on the network. When doing the C++ coding by hand, it is necessary to create all of the code that associates the agent application with the real resources on the network.

TMN++ API

This application programming interface (API) was developed by the Network Management Forum (NMF) to help ease the agent development process. The API allows the application to move from one Q3 interface to another without difficulty. This is useful for developers who must be able to use the same agent applications over different types of Q3 interfaces or over Q3 interfaces from different vendors.

The TMN/C++ API is made up of three APIs: the GDMO/C++ API, the common management information services (CMIS)/C++ API, and the ASN.1/C++ API. The GDMO/TMN++ API allows mapping between GDMO object classes and C++ classes. It provides services to instantiate objects and services to maintain the management information tree (MIT). High-level protocol services are provided by the CMIS/C++ API, which is an interface to common management information services element (CMISE) and association control service element (ACSE). The ASN.1/C++ API lets the developer move data between application-specific data types and ASN.1 data types. It performs operations on ASN.1 data types and can encode and decode data for transmission between systems. Using this API will speed development if one is coding by hand.

8. Testing the Agent Software

As in every step of the agent development process, the agent model is crucial. When testing, it is necessary to have a piece of software that looks like a manager. This manager must be able to generate requests to the agent, and it must understand the agent model. If testing the agent manually, it is necessary to write this mock manager.

To test the agent, a separate test must be created for each MO instance. These tests must send good data to the MO and see if the MO responds properly. Also, the tests must send faulty requests, such as values that the MO should not be able to handle, to see if the tests fail. Because a test must be created for every instance, the number of tests that need to be generated for just one agent application can easily number in the thousands. If an instance is added or the model is changed, it is necessary to create new tests accordingly.

Tools for agent testing can automate this process by providing a mock manager that can interact with the agent model. Then a user interface can be used to create tests using tools. Thus, the thousands of tests that would have to be created by hand can be generated automatically based on the model.

9. Conclusions

Though creating an embedded TMN agent by hand is a daunting task, there are tools that exist to automate part or all of the process, making embedded TMN agent development much simpler and more manageable. These agents are used in the real telecommunications world and are employed on many different kinds of devices, from synchronous optical network (SONET) multiplexers to central office (CO) switches. The growing number of tool kits available has made them a more viable option than using a mediation device, with decreased cost and increased features, benefits, and performance.

Self-Test

1. What is an embedded TMN agent?
 - a. an application that runs on a network element that provides one or more management systems the ability to manage the network element
 - b. an application that uses a manager to manipulate real resources in an embedded environment
 - c. an application that a manager application uses to access its managed resources on a network in a UNIX mediation device
 - d. an application that creates a management information base (MIB)
2. Where in the TMN pyramid does an embedded TMN agent lie?
 - a. network management layer
 - b. element management layer
 - c. network element layer
 - d. service management layer
3. An object is defined as which of the following?
 - a. a variable

- b. a software package that contains a collection of related methods and variables
 - c. an instance of a particular class, where the method of the object is specified by its class and the value of the object is defined by its instance
 - d. both b and c
4. A containment tree is different from a MIB because of which of the following?
- a. MIB is bigger.
 - b. There is no difference—they are essentially the same.
 - c. The MIB represents managed objects possible on a network, while a containment tree holds the real resources on the network.
 - d. The containment tree allows for shorter development time.
5. Which of the following is the specification language used to create an MIB?
- a. TMN
 - b. OSI
 - c. CMIP
 - d. GDMO
6. TMN is defined as which of the following?
- a. a set of standards for all networks
 - b. a type of software
 - c. a set of international standards for the management of telecommunications networks
 - d. a programming language
7. The communications protocol used for communication between manager and agent is which of the following?
- a. SNMP
 - b. TMN

- c. CMIP over an OSI stack
 - d. ASN.1
8. The code for an embedded TMN agent is usually written in which of the following
- a. GDMO
 - b. ASN.1
 - c. C++
 - d. PASCAL
9. Tool kits can be used to do all but which one of the following?
- a. create an agent model in GDMO
 - b. attach an agent application to the real resources on a network
 - c. port to selected RTOS
 - d. generate the C++ code for the agent
10. Which of the following is a feature of an embedded system?
- a. can use erasable, programmable read-only memory (EPROM)
 - b. can use flash memory
 - c. is memory-constrained
 - d. all of the above

Correct Answers

1. What is an embedded TMN agent?
- a. an application that runs on a network element that provides one or more management systems the ability to manage the network element**
 - b. an application that uses a manager to manipulate real resources in an embedded environment
- (See Definition).

- c. an application that a manager application uses to access its managed resources on a network in a UNIX mediation device
 - d. an application that creates a management information base (MIB)
2. Where in the TMN pyramid does an embedded TMN agent lie?
- a. network management layer
 - b. element management layer
 - c. network element layer**
 - d. service management layer

(See [Topic 1](#)).

3. An object is defined as which of the following?
- a. a variable
 - b. a software package that contains a collection of related methods and variables
 - c. an instance of a particular class, where the method of the object is specified by its class and the value of the object is defined by its instance
 - d. both b and c**

(See [Topic 4](#)).

4. A containment tree is different from a MIB because of which of the following?
- a. MIB is bigger.
 - b. There is no difference—they are essentially the same.
 - c. The MIB represents managed objects possible on a network, while a containment tree holds the real resources on the network.**
 - d. The containment tree allows for shorter development time.

(See [Topic 4](#)).

5. Which of the following is the specification language used to create an MIB?
- a. TMN
 - b. OSI
 - c. CMIP
 - d. GDMO**

(See Topic 5).

6. TMN is defined as which of the following?
- a. a set of standards for all networks
 - b. a type of software
 - c. a set of international standards for the management of telecommunications networks**
 - d. a programming language

(See Topic 1).

7. The communications protocol used for communication between manager and agent is which of the following?
- a. SNMP
 - b. TMN
 - c. CMIP over an OSI stack**
 - d. ASN.1

(See Topic 1).

8. The code for an embedded TMN agent is usually written in which of the following
- a. GDMO
 - b. ASN.1
 - c. C++**
 - d. PASCAL

(See Topic 7).

9. Tool kits can be used to do all but which one of the following?

a. create an agent model in GDMO

b. attach an agent application to the real resources on a network

c. port to selected RTOS

d. generate the C++ code for the agent

(See Topic 7).

10. Which of the following is a feature of an embedded system?

a. can use erasable, programmable, read-only memory (EPROM)

b. can use flash memory

c. is memory-constrained

d. all of the above

(See Topic 3).

Glossary

ACSE

association control service element

ASN.1

abstract syntax notation one

API

application programming interface

CD

compact disc

CMIP

common management information protocol

CMIS

common management information services

CMISE

common management information services element

CO

central office

EPROM

erasable, programmable, read-only memory

GDMO

guidelines for the definition of managed objects

GETS

government emergency telecommunications service

GUI

graphical user interface

IPC

interprocess communications

MIB

management information base

MIT

management information tree

MO

managed object

MPU

management processing units

NE

network element

NMF

Network Management Forum

NVRAM

nonvolatile RAM

OAM&P

operations, administration, maintenance, and provisioning

OSI

open systems interconnection

PDU

protocol data unit

RAM

random-access memory

ROM

read-only memory

RTOS

real-time operating system

SCU

shelf control unit

SET

secure electronic transaction

SMF

single mode fiber

SNMP

signaling network management protocol

SONET

synchronous optical network

TL1

transaction language 1

TMN

telecommunications management network