UML Specification of a Generic Model for Fault Diagnosis of Telecommunication Networks

Armen Aghasaryan[1], Claude Jard[2], and Julien Thomas[3]

[1] Alcatel Research & Innovation, Route de Nozay, 91461 Marcoussis, France
Armen.Aghasaryan@alcatel.fr
[2] IRISA/ENS Cachan, Campus de Ker-Lann, 35170 Bruz, France
Claude.Jard@irisa.fr
[3] IRISA/INRIA, Campus de Beaulieu, 35042 Rennes, France

Abstract. This document presents a generic model capturing the essential structural and behavioral characteristics of network components in the light of fault management. The generic model is described by means of UML notations, and can be compiled to obtain rules for a Viterbi distributed diagnoser.

1 Introduction

This paper presents the results of the continued efforts on generic modeling initiated within the Magda projects [1] and [2]. The generic model captures the essential structural (generic components and their relations) and behavioral (interactions between the generic components) characteristics of telecommunications network components in the light of their utilization in fault management tools. The generic model covers both circuit-based and packet-based networks, despite of divergent approaches adopted by the respective standardization bodies. The generic model is described by means of UML notations, namely, class diagrams, sequence diagrams and instance diagrams. These diagrams are intended to be used in 1/ derivation of the technology-specific models and 2/ generation of rules on generic component instances. Although the targeted management applications work with the derived specific models, but in certain cases they can directly apply the rules defined on generic components. Thus, the effort necessary for deriving a specific model is significantly reduced. For the diagnosis application we have considered, we proved that the generic model can be compiled to obtain generic rules.

2 Basic Concepts

2.1 Structure

The basic concepts of our generic model are guided by the ITU-T standard Generic Functional Architecture of Transport Networks [6]. The layering concept in this architecture introduces client-server relations between the adjacent layer networks, while the partitioning concept allows the decomposition of a layer network into subnetworks and links. The end-to-end connectivity in the server layer, network connection, is obtained by concatenation of link connections. A *trail* in the server layer provides the communication service between two neighboring nodes of the client layer network, see Fig. 1a. In the case of multiplexing, it carries several *link connections* of a client network in the server network; we speak also of a *containment* relation be-

The presented results have been funded by the French national RNRT Magda2 project on fault management.

tween the respective Connection Termination Point (CTP) and Trail Termination Point (TTP) managed entities of the neighboring network layers. The CTP/TTP entities of the same network layer are connected, as shown in the figure, through *matrices* and subnetworks.

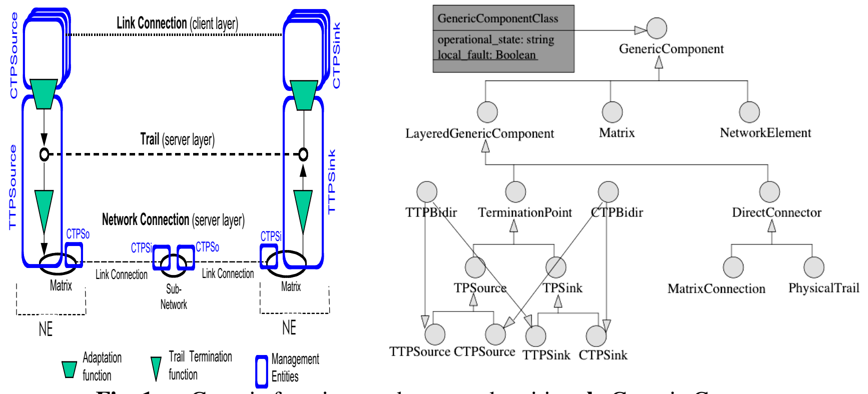


Fig. 1. a. Generic functions and managed entities. .**b.** Generic Components.

These concepts are largely adopted in circuit-based technologies and they are represented in object-oriented management information models [5]. In packet-based technologies however these generic concepts are not explicitly expressed and the information models are often based on SNMP table structures. Nevertheless, it can be shown that these concepts remain pertinent also for packet-based connection-oriented networks, and the respective objects can be extracted from table-based information models. In MPLS [8], the FEC (Forwarding Equivalence Class) aggregation approximates the termination function and can be seen as a TTP, while the label assignment is related to the adaptation function and can be represented by a CTP.

2.2 Fault Behavior

Due to their physical nature and the associated monitoring mechanisms, the main role in fault propagation across the network is played by transmission failures. A transmission failure is physically propagated through the network: horizontally, along a network connection at a given layer network, and vertically, through the higher layer networks. In addition, the detected faults can be propagated via *monitoring signals.* So, once a failure/degradation is detected at a TTPSink, failure indications are sent to the corresponding CTPSink objects on the client layer. Further, this information can be forwarded downstream along the network connection in the client layer until the respective TTPSink is reached (*Forward Defect Indication).* Note that the propagation can be interrupted whenever one of the objects on the propagation path is in a “non-communicating” (e.g. Disabled) state (*alarm masking*).

3 Structural relations

3.1 Generic Components package

We use an abstract notion of *GenericComponent* to represent any network component that can be faulty and/or can participate in a fault propagation, see Fig.1b. The generic components are interrelated by means of peering and containment relations. In order to allow working with a programming language (e.g. Java) which does not support multiple inheritance, all the generic components are declared as Interfaces. However, the state variables can not be declared as attributes of an Interface, this is why we define them in an additional class GenericComponentClass and impose (informally) that in a specific model any class implementing a generic component must inherit from this class to obtain the uniform definition of state variables. We identify 4 main classes of generic components : *NetworkElement* represents a physical network element (device); *Matrix* represents a logical or physical component that regroups a set of cross-connections (matrix connections); *TerminationPoint* represents a logical or physical component used in transmission; *DirectConnector* represents a matrix connection and physical link components, the latter is a trail at the lowest layer network.

3.2 Layer Network (partitioning) package

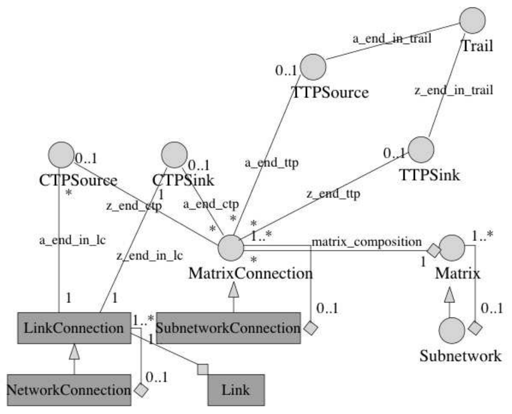


Fig. 2. Peering relations : layer network (horizontal) partitioning.

The diagram of Fig. 2. defines the horizontal (or peering) relations between the generic components. *MatrixConnection* class represents a cross-connection between two termination points on the same network element. *LinkConnection* class represents a logical (*indirect*) connection between two CTP objects on two neighboring network elements. *Link* class represents a list of link connections grouped together for usage in routing algorithms. *Trail* class represents a logical (*indirect*) connection between two TTP objects on two distant network elements, it symbolizes a transport service provided by a given layer network to its client layer network. The *Client-Server Relations package* is used to define the client-server (vertical) layering as containment relations between *CTPSource* and *TTPSource* (source aggregation), and between *CTPSink* and *TTPSink*. The *Physical Location package* introduces another type of containment relations: containment by physical location. A network element contains a matrix, and a number of physical ports. The *NE view package* is aimed at presenting a local view of the network hierarchy. It proposes a structure of recursively embedded

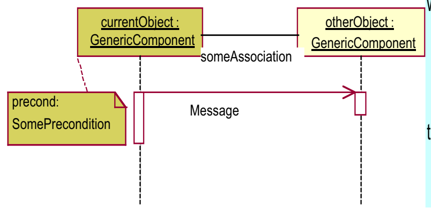
Layers where each layer assembles the local termination points belonging to the given layer network. This view is important if one aims at automatic model discovery from the information available in the network devices. In that case, a complete model instance is constructed having as an input the entities and dependencies described in this package.

4 Fault Dynamics

In order to describe the fault-related behavior of model components the notion of tile was introduced in [4]. The tiles are composed of a pre-condition part - conditions on attribute values and reception of messages; an action part - sending of messages, possibly under some conditions, and a post-condition part - new attribute values. The tiles can be easily described with a rule script and can be directly called in fault management applications that make use of a rule engine. We introduce an equivalent UML compliant description mechanism which allows to associate fault-related behaviors with the generic components and to transform them into a rule script.

4.1 Rule description with UML diagrams: the concept

The main idea is to describe a rule by means of one sequence diagram and one instance diagram. The sequence diagram represents the *message exchange* between the class instances and defines the *conditions* on attributes in the associated comment blocks. On the other hand, the instance diagram specifies the *structural relations* between the class instances. The instance diagram will therefore introduce new conditions to be verified in the rule. Fig. 3 shows such a pair of sequence and instance diagrams. The corresponding rule script (in Ilog JRules syntax) generated with default conventions is shown on the right.


currentObject : GenericComponent
otherObject : GenericComponent
someAssociation
precond: SomePrecondition
Message

```
when {
    ?currentObject : GenericComponent
    ?otherObject : GenericComponent
    evaluate (?currentObject.is_associated(SomeAssociation, ?otherObject))
    evaluate (SomePrecondition)                    }
then {
    ?otherObject.receive(new Message())                }
```

Fig. 3. A sequence diagram with an instance diagram describes a rule**.**

We use this mechanism for defining *generic tiles*, i.e. behaviors associated with generic components such that the corresponding rule script is readily applicable, by inheritance, to the respective components in a specific model. Of course, the same mechanism can be applied during the definition of the specific model in order to describe supplementary specific fault behaviors.

4.2 Horizontal and vertical propagation

Propagations between the components of the same network layer can be in *forward* or *backward* directions. In either direction, they can follow *direct connections* (matrix connections) or *indirect connections* (link connections, trails). For the sake of briefness, we chose to present the case FDI-1 of direct connections.

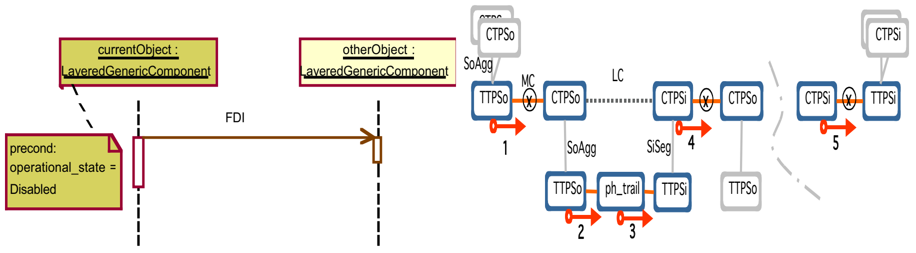

currentObject : LayeredGenericComponent
otherObject : LayeredGenericComponent
FDI
precond: operational_state = Disabled
CTPSo
SoAgg
TTPSo
MC
CTPSo
LC
CTPSi
CTPSo
CTPSi
CTPSi
TTPSi
1
SoAgg
SiSeg
4
5
TTPSo
ph_trail
TTPSi
TTPSo
2
3

Fig. 4. Forward Propagation via a direct connection**.**

The sequence diagram of Fig. 4 defines a behavior where a generic component with *operational_state = Disabled* communicates a message FDI to another generic component. The right part summarizes the 5 cases where such a behavior can happen: via matrix connections or physical trails. The concerned objects are outlined with blue rectangles, the associations between them are indicated with red lines, the *currentObject* is indicated with a circle where an arrow originates. Propagation and/or Masking is a behavior inherent to all the layered generic components and can be easily modeled by pre-conditioning the forwarding of messages from an upstream object to an downstream object, by the test of the *operational_state* of the intermediate object. Vertical propagation is modeled as a message exchange between two generic components with a precondition on the operational state of the object sending the message. The idea behind it is that any generic component in a faulty state will automatically propagate its state to all the components it contains. So, this behavior is essentially based on the containment relation.

5 Model Compiler

Using the class and deployment diagrams, our tool builds the actual objects and their associated links, in order to obtain the model instance of the specific network to be supervised. These objects will be used by the diagnosis application. On the other hand, the sequence diagrams are compiled to produce the rules (expressed in Jrules) needed by the diagnoser. The OSCAR prototype has been connected to the Objecteering case tool using the XMI interface. Precisely, OSCAR takes in input a UML model and produces a set of Java classes, a set of rules (an XML file describing a set of condition/action) and a configuration file, as illustrated in Fig. 5. Another step of facilitation of the modeling task consists in the discovery of network topology and the respective instantiation of the correlation model used by diagnosis modules. The automatic instantiation can be applied to the structural part of the model by inspecting the supervised network elements and by extracting the connectivity information contained in the SNMP MIB tables. This information then is mapped onto an object structure of a specific model for MPLS networks derived from our generic model which in its turn is compliant with ITU-T G.805 Recommendations. As a result of this mapping one generates a set of XML files representing the logical and physical topology of the supervised network in the terms comprehensible for diagnoser modules. The same schema can be used in order to take into account the network reconfigurations that may happen after the initial model instance was communicated to diagnoser modules. This functionality is integrated within a generic Topology Manager tool developed in Alcatel R&I Lab. In our experiment, the distributed alarm correlation task is performed by a collection of Viterbi Diagnoser (VD) modules. Each of these modules is in charge of a limited part of the network, typically one network element . The task of a VD is to collect alarms produced by the region it supervises, and to recover all behaviors of the supervised region that could explain these alarms [3]. The whole chain of our model-based approach to diagnosis have been demonstrated on the Alcatel management platform ALMAP.

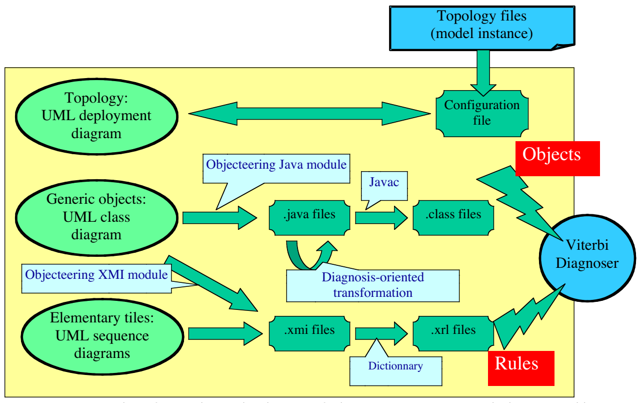


Fig. 5. The functional view of the OSCAR model-compiler

References

[1] MAGDA2-GMPLS Architecture, A. Aghasaryan, F. Touré, A. Benveniste, S. Haar, E. Fabre, F. Krief, Magda2 project deliverable MAGDA2/HET/LIV/1, November 30, 2002.

[2] Modeling Fault Propagation in Telecommunications Networks for Diagnosis Purposes, A. Aghasaryan, C. Dousson, E. Fabre, A. Osmani, and Y. Pencolé, XVIII World Telecommunications Congress, Paris, 22-27 September 2002.

[3] Algorithms for Distributed Fault Management in Telecommunications Networks, E. Fabre, A. Benveniste, S. Haar, C. Jard, A. Aghasaryan, This conference.

[4] Fault detection and diagnosis in distributed systems : an approach by partially stochastic Petri nets, A. Aghasaryan, E. Fabre, A. Benveniste, R. Boubour, and C. Jard, Journal of Discrete Event Dynamic Systems, Kluwer Academic Publishers, Boston, Vol.8, no.2, June 1998.

[5] Generalized Multiprotocol Label Switching: An Overview of Signaling Enhancements and Recovery Techniques, Ayan Banerjee, John Drake, Jonathan Lang, Daniel Awduche, Lou Berger, Kireeti Kompella, and Yakov Rekhter, IEEE Communications Magazine, July 2001.

[6] ITU-T G.805 Generic Functional Architecture of Transport Networks, March 2000.

[7] ITU-T G.782 Types and general characteristics of synchronous digital hierarchy (SDH) equipment, 1994.

[8] Multiprotocol Label Switching (MPLS) FEC-To-NHLFE (FTN) Management Information Base, Internet Draft, draft-ietf-mpls-ftn-mib-09.txt, October 2003.