Algorithms for Distributed Fault Management in Telecommunications Networks*

Eric Fabre, Albert Benveniste, Stefan Haar[1], Claude Jard[2], and Armen Aghasaryan[3]

[1] IRISA/INRIA, Campus de Beaulieu, 35042 Rennes cedex, France
[2] IRISA/ENS-Cachan Campus de Ker-Lann; 35042 Rennes cedex, France
[3] Alcatel Research & Innovation, Next Generation Network and Service Management Project, Alcatel R&I, Route de Nozay, Marcoussis, 91461 France

Abstract. Distributed *architectures* for network management have been the subject of a large research effort, but distributed *algorithms* that implement the corresponding functions have been much less investigated. In this paper we describe novel algorithms for model-based *distributed* fault diagnosis.

1 Introduction

Distributed self-management is a key objective in operating large scale infrastructures. Fault management is one of the five classical components of management, and is the subject of our work. In this paper, we consider a distributed architecture in which each supervisor is in charge of its own domain, and the different supervisors cooperate at constructing a set of *coherent* local views for their respective domains. By coherent, we mean that the different views for the different supervisors agree on the interfaces of their respective domains. Of course, to ensure that the approach can properly scale up, the corresponding global view should never be actually computed.

Fault diagnosis has been addressed by different means. Historically, the first approach was by means of rule-based expert systems [10,11]. Such systems are generally limited to simple correlation rules, with small scope, and hardly capture all the complexity of the network reaction to a failure. Connexionist and related learning techniques from Artificial Intelligence area have been investigated to avoid the burden of providing detailed and explicit knowledge on the supervised domain [5]. But again, such methods are not scalable, and difficult to update when the network evolves. To avoid these drawbacks, the main trend nowadays is in favor of model based approaches [8,7,9]. Most of them rely on topological information, both physical and logical (this information is easy to get by scanning Management Information Bases). Using the topology of interactions, successive correlations with an observed symptom can be traced back to their possible alternative causes.

* This work was supported by the French National RNRT project MAGDA2, funded by the Ministère de la Recherche.

The present approach goes further in this direction. First of all in the size and details of the model (see fig. 1) : the physical and logical topologies (Network Elements + connections) are modeled as a graph of interconnected Managed Objects (MO), and each MO is itself described as a small *dynamic* system. In the management plane, MOs are equiped with their own fault management function and are responsible of the production of alarms. Alarms are emitted when a local failure is detected, or in reaction to the failure of a neighboring MO that was necessary to guarantee a correct service. This second phenomenon is at the origin of failure propagations in the network. We refer the reader to [1] for details on the modeling, and for the description of a tool that automatically builds the model. Second key feature of our approach : fault diagnosis is performed in a *distributed* way. Specifically, we assume alarms are not sent to a global supervisor, but are rather collected by local supervisors, in charge of part of the system. In this scenario, a local supervisor only needs to know the local model of its domain (for example one NE in fig. 1), and cooperates asynchronously with supervisors of neighboring domains. We believe this is the key to cross-domain fault management.

For space reasons, we only give a glance at the theory behind distributed diagnosis algorithms, by means of a toy example (see [2,3,4] for details).

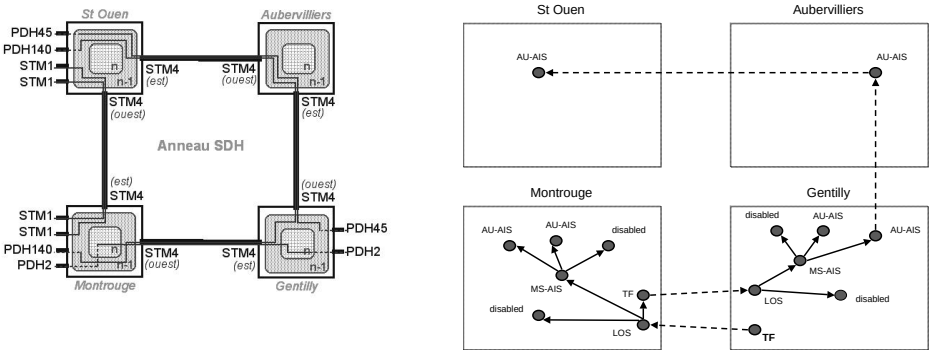

Fig. 1. Left : Part of the Paris area SDH/SONET ring (topological view, at the Network Element level). In the model of this network, each NE is further decomposed into Managed Objects, corresponding to the SDH hierarchy (SPI, RS, MS, etc.). Right : a failure scenario on this model, with propagation of the failure, entailing correlation in the alarms raised.

2 A structure to represent runs with concurrency

A central feature in large distributed dynamic systems is that many components run in parallel, so several events can occur at the same time, *i.e.* "concurrently." It is therefore crucial to represent runs of such systems in a framework that captures this independence, and sequences of events are definitely inappropriate for that. Since the components of our network are finite state machines, this

suggests to use safe Petri nets to model components, and to represent runs of the network (*i.e.* failure propagations) with so-called "true concurrency semantics." We illustrate this on a toy example, given in fig. 2.

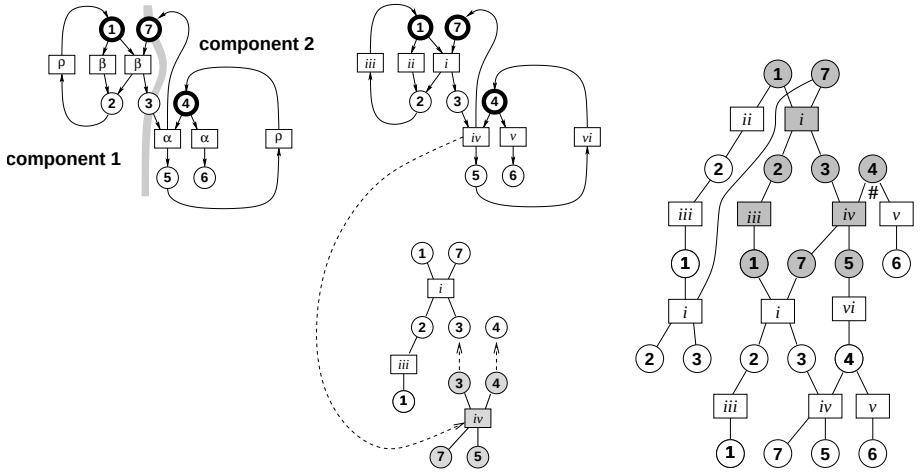

Fig. 2. Running example in the form of a Petri Net, viewed as two components interacting through shared places 3 and 7 (left). Thick places indicate the initial marking. Right : representation of runs of this system under the form of a branching process. "Time" goes from top to bottom.

On the example, places 1 and 2 represent states *ok* or *down* of component 1 (with a possibility of self-repair by transition *iii*). Component 1 can thus go down in two ways, through *ii* or *i*. The second possibility fills place 3 which models a propagation of the failure to component 2 : place 4 corresponds to state *ok*, place 5 to a temporary failure (self-repair possible by *vi*), and place 6 to a definitive *down* state. Notice that, on the leftmost right picture of the model, transitions are labeled by α, β, ρ which corresponds to the *alarms* produced when they fire.

The mechanism of constructing a run of the Petri net $\mathcal{P}$ in the form of a partial order is illustrated in the 2nd and 3rd diagrams. Initialize any run of $\mathcal{P}$ with the three conditions labeled by the initial marking $(1, 7, 4)$. Append to the pair $(1, 7)$ a copy of the transition $(1, 7) \to i \to (2, 3)$. Append to the new place labeled 2 a copy of the transition $(2) \to iii \to (1)$. Append, to the pair $(3, 4)$, a copy of the transition $(3, 4) \to iv \to (7, 5)$ (this is the step shown). We have constructed (the prefix of) a run of $\mathcal{P}$, where concurrency is explicitly displayed : the ordering of events labeled *iii* and *iv* is left unspecified, we only know they occur after (or as a consequence of) *i*.

Now, all runs of $\mathcal{P}$ can be constructed in this way. Different runs can share some prefix. The rightmost diagram shows a *branching process* (BP) of $\mathcal{P}$, obtained by superimposing the shared parts of different runs. The gray part is a copy of the run shown in the middle. The alternative run on the extreme left

of this diagram (involving successive transitions ii, iii, i) shares only its initial places with the run in gray. On the other hand, replacing, in the gray run, the transition labeled iv by the one labeled v yields another run. We say there is a *conflict* at place 4 : a choice must be made between firing iv or v. A conflict takes place each time a place is branching in this diagram. Branching processes thus encodes sets of executions of a PN in a compact manner : a run of the PN corresponds to selecting part of the BP which is both conflict free and causaly closed. Observe that by nature a BP has no oriented cycle. The maximal BP of a Petri net $\mathcal{P}$ is called its *unfolding* : $\mathcal{U}_\mathcal{P}$. Places and transitions of a BP are rather called *conditions* and *events*, and are labeled by places and transitions of the original net $\mathcal{P}$.

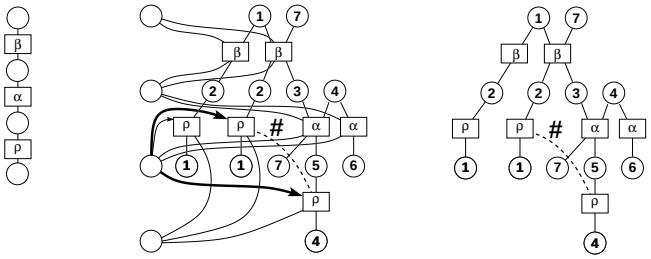

Fig. 3. Left : an alarm pattern $\mathcal{A}$. Center : the product $\mathcal{A} \wedge \mathcal{U}_\mathcal{P}$. right : same net without the conditions of $\mathcal{A}$.

Assume the system modeled by $\mathcal{P}$ runs and that labels (alarms) produced by its transitions are collected by a sensor. We represent the collected alarm pattern under the form of a net $\mathcal{A}$ (left diagram on fig. 3). The (centralized) diagnosis problem can be formalized in the following manner : compute all runs of $\mathcal{P}$ that could explain the observed alarm pattern $\mathcal{A}$. Deciding what failure actually occured is then a post-processing on this set of possible runs. Formally, this amounts to computing all runs of the product net $\mathcal{A} \times \mathcal{P}$, which is $\mathcal{P}$ *constrained* by observations $\mathcal{A}$. Equivalently, we want $\mathcal{U}_{\mathcal{A}\times\mathcal{P}} = \mathcal{A} \wedge \mathcal{U}_\mathcal{P}$, where $\wedge$ is a special product on branching processes, designed to avoid oriented cycles. The result is depicted on fig. 3, and can be built recursively, in an asynchronous manner, just like branching processes of $\mathcal{P}$. Observe that an extra conflict relation is created (dashed line labeled with #) to capture the fact that alarm ρ can be explained either by (2) $\rightarrow$ iii $\rightarrow$ (1) or by (5) $\rightarrow$ iii $\rightarrow$ (4). Possible explanations to $\mathcal{A}$ correspond to maximal runs of $\mathcal{A} \wedge \mathcal{U}_\mathcal{P}$ that are labeled by *all* alarms of $\mathcal{A}$ (some runs of $\mathcal{A} \wedge \mathcal{U}_\mathcal{P}$ may only explain the first alarms and then stop).

3 Distributed diagnosis with two supervisors

This is the main novelty of this paper. Assume a local sensor collects alarms produced component i, under the form of a local alarm pattern $\mathcal{A}_i$, $i = 1, 2$. The objective is to build a *distributed* view of $\mathcal{U}_\mathcal{P} \wedge \mathcal{A}_1 \wedge \mathcal{A}_2$. Alarm patterns are processed by two local supervisors communicating asynchronously.

Fig. 4 (top) displays the information available to supervisor i : $\mathcal{A}_i$ together with the local model of component i. The bottom diagrams represent the local views of the diagnosis. They correspond to projections on events of each component of $\mathcal{U}_\mathcal{P} \wedge \mathcal{A}_1 \wedge \mathcal{A}_2$. Specifically, on the side of component 1, one has transitions of component 1 (explaining local alarms $\mathcal{A}_1$) plus abstractions of the action of component 2 on shared places 7 and 3. These events appear as empty rectangles. To recover a complete trajectory of $\mathcal{P}$, one has to glue matching trajectories on each side. This corresponds for example to gluing events indicated by arrows on the picture, to obtain the grey trajectory of $\mathcal{P}$. Observe that this distributed representation of the diagnosis is more efficient than the centralized one : to be able to glue two local runs, it is enough that they have maching behaviours *only on shared places*. So, for a given history of these shared places, m trajectories of component 1, and n trajectories of component 2 actually correspond to $m \times n$ trajectories of $\mathcal{P}$.

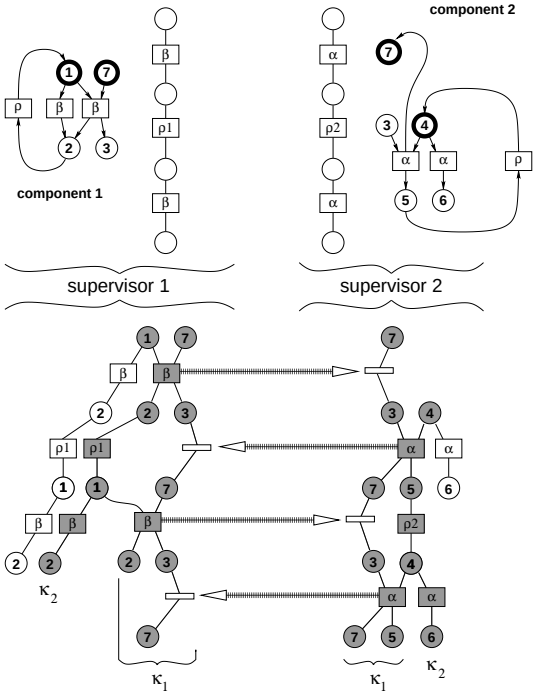

Fig. 4. Distributed diagnosis: two coherent local views of the unfolding $\mathcal{U}_\mathcal{P} \wedge \mathcal{A}_1 \wedge \mathcal{A}_2$ are built by two supervisors cooperating asynchronously.

As for the centralized case, the distributed view of the diagnosis can be obtained recursively, but requires communication between supervisors. Each arrow actually corresponds to the communication of a new event to append. However, the cooperation between the two supervisors need only asynchronous communication. Each supervisor can simply "emit and forget." Diagnosis can progress

concurrently and asynchronously at each supervisor. For example, supervisor 1 can construct the branch $[(1) \rightarrow \beta \rightarrow (2) \rightarrow \rho_1 \rightarrow (1) \rightarrow \beta \rightarrow (2)]$ as soon as the corresponding local alarms are collected, without ever synchronizing with supervisor 2.

4 Conclusion

We have proposed an unfolding approach to the distributed diagnosis of concurrent and asynchronous discrete event dynamical systems. Our presentation was informal, based on a toy illustrative example. The related mathematical material is found in [3]. A prototype software implementing this method was developed at IRISA. This software was subsequently deployed at Alcatel on a truly distributed architecture, no modification was necessary to perform this deployment. Model-based approaches suffer from the burden of getting the model. We have developed a tool [1] that automatically generates the needed model. This work is also presented at this conference.

References

1. A. Aghasaryan, C. Jard and J. Thomas : UML specification of a genericmodel for fault diagnosis of telecommunications networks. ICT 2004.
2. A. Benveniste, E. Fabre, C. Jard and S. Haar : Diagnosis of asynchronous discrete event systems, a net unfolding approach. IEEE Trans. on Automatic Control, 48(5), 714–727, may 2003.
3. A. Benveniste, S. Haar, E. Fabre and C. Jard : Distributed monitoring of concurrent and asynchronous systems. In *Proc. of CONCUR'2003,* Sept. 2003. See also IRISA report No 1540, http://www.irisa.fr/bibli/publi/pi/2003/1540/1540.html
4. E. Fabre. Monitoring distributed systems with distributed algorithms. In Proc of the 2002 IEEE Conf. on Decision and Control, 411–416, Dec. 2002, Las Vegas, 2002.
5. Bennani Y. and Bossaert F : Modular Connectionist Modeling and Classification Approaches for Local Diagnosis in TelecommunicationTraffic Management. *Int. J. of Computational Intelligence and Applications,* World Scientific Publishing Company.
6. J. Engelfriet : Branching Processes of Petri Nets. Acta Informatica 28, 1991, pp 575–591.
7. B. Gruschke : Integrated Event Management: Event Correlation using Dependency Graphs. In Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management 1998, DSOM 98, Newark, DE, USA, October, 1998.
8. S. Kaetker and K. Geihs : A Generic Model for Fault Isolation in Integrated Management System. Journal of Network and Systems Management, Special Issue on Fault Management in Communication Networks, Vol. 5, No. 2, June 1997.
9. S. Kliger, S. Yemini, Y. Yemini, D Ohsie, and S. Stolfo. A coding approach to event correlation. In IFIP/IEEE Int. Symp. on Integrated Network Management IV, 1995, 266-277.
10. J. Liebowitz, editor. Expert System Applications to Telecommunications. John Wiley and Sons, New York, NY, USA 1988.
11. Y. A. Nygate : Event correlation using rule and object based techniques. In IFIP/IEEE Int. Symp. on Integrated Network Management IV, 1995, 278-289.