A Model Checking Approach to Network Fault Management

Hesham Hallal, Alexandre Petrenko, Sergiy Boroday
CRIM,
550 Sherbrooke West, Suite 100,
Montreal, H3A 1B9, Canada,
{Hallal, Boroday, Petrenko}@crim.ca

and

Andreas Ulrich
Siemens AG, Corporate Technology SE1,
Otto-Hahn-Ring 6, 81739 München, Germany,
Andreas.Ulrich@siemens.com

ABSTRACT

We propose an approach to analyze network systems based on formal methods. The trace analysis approach relies on model checking and ensures an automated and exhaustive analysis of traces of executions collected from networks through monitoring. In particular, we show how to apply the approach to the fault management procedure, and we illustrate using an example.

Keywords
Model Checking, Networks, Fault Management, Trace Analysis, Property Checking.

1. Introduction

Maintaining correct operation of a computer network is highly intricate. In this age of globalization, where a network could easily span the glob, the need for network management (at the various layers) becomes even greater, especially with networks that include multi-vendor and multi-protocol elements and clusters. This has yielded several attempts to standardize the management interfaces of networks in order to alleviate the associated complexity (a formidable knowledge of the different networks and a costly learning curve). The Telecommunication Management Network (TMN) standards, developed in the early 90's by the ITU, define the main areas associated with the management concept. Fault management, Configuration management, Account management, Performance management, and Security management are often considered a good definition of what management activities should encompass. Examples of these activities include, but are not restricted to the following: responding to faults when a network element breaks down, addition or removing subscribers, changing network configuration, updating software of network elements, preventing attacks and detecting intrusions, congestion control.

While standardization helps reducing the complexity managing network systems, it stops short from solving all the aspects of the problem. Actually as networks grow larger, the information available for processing, in the course of a management task, becomes literally unmanageable even when it is uniformly represented. This adds to the complexity of the validation and debugging problems of networks, and thus to the overall cost. In the last decade, there have been numerous attempts to automate and simplify the task of managing networks. The approach that is considered in the TMN layered model of network management, where the management tasks are distributed on four levels (The Element management, The Network management, The Service management, and The Business management), represents a major step in modularizing the tasks and distributing the load. Based on this model, numerous attempts to automate the management process on the different layers have been made [1, 16, 9]. For example, [10] reports the following attempts to deal with the alarm correlation problem of the fault management process associated with the network management layer: Rule-based correlation, Fuzzy logic based correlation, Model-based reasoning, Filtering, Event forwarding discrimination, etc.

Meanwhile, research results, mainly in the academia, continue to build a strong belief that the use of formal methods would contribute to a drastic reduction in the complexity and cost of the development of distributed applications in general. Indeed, once a formal specification of the behavior of a system becomes available, many development activities, such as validation and test derivation could easily be automated. Based on these expectations, a number of tools that support formal specifications (using languages such as SDL or UML) and modeling activities have recently become available. However, to fully benefit from formal approaches, a company should reorganize the whole engineering process, its personnel has to be accordingly trained, so inevitably, there is a costly learning curve that not every company is ready to absorb. The reality is that formal models are rarely produced in practice from requirement specifications, except for certain safety critical applications that fully justify a new development discipline.

Nevertheless, formal approaches could still be used to automate important steps of the development process

while not requiring that developers start producing formal specifications or even learn much about formal languages. Instead, by relying on ad-hoc observation facilities or standard monitoring tools, qualified personnel would be able to discern manually in a picture any pattern of events that indicates the existence of a problem or a fault in what represents the execution of the system over a period of time. This is exactly where we think the automation is required. We believe that formal approaches make it possible to automate the analysis of distributed systems based on collected logfiles and to reduce, if not eliminate, the error margin associated with manual analysis.

In a previous work [7, 8], we have developed an approach to automatically verify (using model checking [3]) whether a distributed system exhibits certain properties based on a formal model of the system determined from a trace of events that happened within the system. A lighter version of this approach, which uses XML transformations to detect properties that do not require model checking, appeared in [11].

We believe that the trace analysis approach can be applied in the field of network management, especially on the Element management and Network management levels of the TMN model. Therefore, in this paper, we show how such a formal approach -based on formal methods and techniques- can be used in the context of fault management at the aforementioned levels of the network model.

This paper is organized as follows: Section 2 overviews the trace analysis approach and its application in testing distributed systems. Section 3 presents the main characteristics of the network management problem, which motivate the application of the trace analysis approach in this context. Section 4 shows, through an example, how the trace analysis approach can be used in the context of fault management. Finally, Section 5 concludes this paper and presents the main potential extension of this work.

2. Trace Analysis of Distributed Systems

A general approach to trace analysis could be outlined as follows. The distributed system is instrumented in a way that events executed by the distributed processes are collected. Such events typically denote the send and receive of messages, local actions and others. A trace is produced that includes all the events collected during a system run, and an appropriate analysis tool can be applied to check the trace against some user-defined properties. A large body of work on developing various tools to visualize traces already exists, see [8] for a summary. The main goal is to facilitate the efforts of the designer or tester for locating and correcting bugs by filtering out unrelated information and by offering a proper visualization of traces. The analysis is performed manually either online (simultaneously with the system execution) or post mortem. Another group of methods targets the analysis phase by offering means to verify certain properties in the distributed system under test (SUT). Practically, when it comes to developing a tool for checking properties in execution traces, one would tend to reuse general-purpose model checkers, normally reliable, highly sophisticated, and versatile analysis tools. Therefore, given that a proprietary monitoring tool provides an event trace that contains sufficient information to analyze certain properties and that events represent by the values of their parameters all the key characteristics of a system under surveillance that are reflected in properties to be verified in the event trace. Then for an available model checker, ObjectGEODE in our case, we had only to develop a front-end tool that produces a formal model and a property description as required by the model checker. The model is written in SDL, the specification and Description Language, and the properties are written in the property specification language GOAL, specific to ObjectGEODE. The results of the verification include a verdict whether the property was met in the model of the system and examples to show the execution sequences that satisfied/violated the property. As a result, the trace analysis approach can be summarized in the diagram in Figure 1.

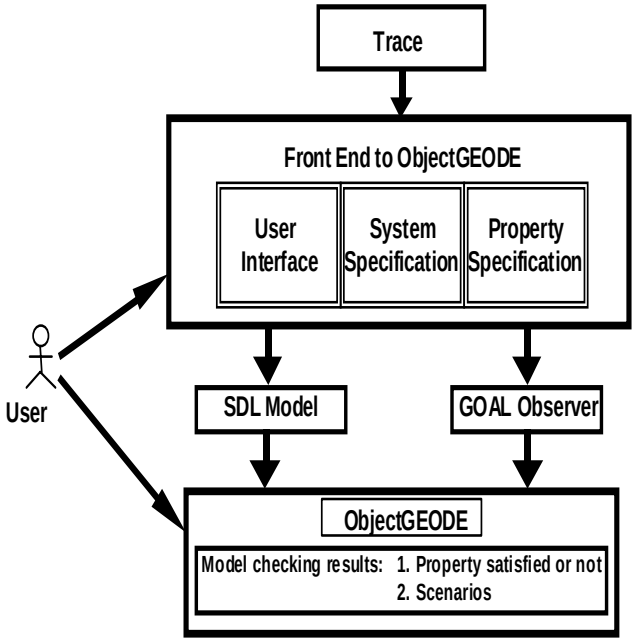

Figure 1: Workflow of the trace analysis approach.

2.1 SDL Model

In detail, an SDL model of the tested system that relies on a given trace reflects the following aspects: structure, behavior, communication, and data. The structure of the system is modeled by the hierarchy: system/block/process/procedure statements in SDL. In our case, it is sufficient to define a system with a single block that is composed of several processes. The overall behavior of the system is modeled by the joint behavior of a set of communicating processes in SDL. These processes correspond to entities of the real system whose behaviors are recorded in the trace; thus making each of the processes linear, as in most cases we cannot identify

any two states of it, so no cycles can be deduced. The representation of a process can be obtained by projecting the collected trace into the set of events it executes: send, receive, and local events and subsequently inserting states in between communication events, while representing local events as SDL tasks. The asynchronous communication between processes is achieved via signals with optional signal parameters (that represent exchanged data) and channels. Input signals to a process are stored in a queue before reading them. Thus, unknown delays in real communication channels of the distributed system are represented by those of input queues, associated to each SDL process. This means that in our framework, the whole communication media of the system is simply modeled with individual queues of SDL processes.

2.2 Property Specification

The property to be checked is expressed as an observer [6] in the GOAL language. A GOAL observer implements in fact a finite automaton with accepting states [2]. GOAL is similar to SDL, but has some syntactic and semantic differences [2]. Observers are usually described in terms of entities (objects, signals etc.) of the tested system, i.e., they can be associated to the SDL system. This makes communication signals directly accessible for observation while other model data, e.g., variables and states, are accessible to observers using probes. Probes represent pointers to SDL entities. In addition, GOAL allows the declaration of two types of designated states: *success* and *error* states. Entering success states (error states) indicates that the system respects (violates) the property expressed in the observer.

2.3 Verification in OG

Once the representation of the distributed system and the property is complete, the ObjectGeode (OG) model checker is used to verify whether the system satisfies the property [14]. This is achieved in the so-called exhaustive simulation mode, when the tool performs all the possible executions of the system and builds a state graph of the SDL system. The state graph represents all the possible interleavings of events in the collected trace. To perform model checking, the tool builds the synchronous product of the observer and the specification of the system. The OG simulator outputs a report of the number of errors and successes encountered, and it presents the scenarios that lead to the observer errors and successes.

2.4 Front-End to ObjectGEODE

The front-end to OG includes two modules:

1. A system specification tool TRAYSIS that builds an SDL specification from the collected trace. This tool automates the process of producing an SDL specification from a logfile of events specified in XML syntax. It supports both synchronous and asynchronous communications and offers the following main features:
   - The tool checks the logfile for flaws that would render the generated SDL model syntactically incorrect, e.g., missing communication events, presence of SDL keywords in the logfile, and the use of some illegal characters.
   - The tool enables the user to cope with large logfiles by offering a combination of filtering techniques including process filtering, signal filtering and segment filtering.
   - The tool offers preliminary analysis of the generated model by presenting listings of the processes, variables, and signals occurring in the model. Such information should be essential to the property specification procedure.
2. A property specification tool called Property Manager[*] that eases the process of writing properties for trace analysis. This tool offers the user a library of predefined patterns [4, 12]. Every pattern of the library is a parameterized template of a GOAL observer. The tool helps the user customize the observer to specific settings of the observed event trace.

3. Applying Trace Analysis to Network Management

In this section, we discuss how the trace analysis approach can be applied in the field of network management, especially in the context of fault management in networks on the Element management and Network management levels of the TMN model.

In general, an element management system (EMS) is usually deployed for a group of Network Elements (NE's) that are of the same type, e.g., DCS or SONET. Normally, an EMS is exposed to the complete management-information content of all the NE's in its domain. Therefore, the tasks performed by the EMS in the context of element management include activities that could be classified in the following main groups [5]:

- Service provisioning (planning, deployment),
- Service assurance (maintenance, monitoring and control),
- EMS and NE operation support (ease of use, remote access),
- Automation enabling (normalization, information storage).

Meanwhile, an EMS is not responsible for all the traffic management related tasks.

Service assurance is the obvious place where we believe the trace analysis approach can ease the job of the operator by automating the process of correlating alarms, detecting malfunctions, and localizing faults. To exemplify the level of complexity that could be associated to these tasks, we consider the simple scenario, where an EMS receives events and alarms from NE's on several ports. In such case, the ordering in which the alarms are received and processed by the EMS affects directly the outcome of any analysis, whether the analysis is manual or automatic. Meanwhile, through monitoring, the executions of the NE's are recorded and logged in event traces. Then, a formal model of the monitored NE group could be generated on which the properties of interest would be verified following the trace analysis approach. Actually, the model checking techniques on which the approach relies eliminate the risk of missing a certain error alarm - by building the graph that encodes all the possible orderings of the recorded events.

[*] We thank Qing Fan for his contributions to the tool.

Properties for this type of analysis can represent operator defined requirements as well as service agreements that are specified on higher levels of the network model but expressed in the NE terminology. Moreover, properties could express intrusion signatures which, when verified in the traces, help detect security breaches to the network system.

Another area where our approach could be applied is the analysis of the EMS's. In some network systems, each EMS oversees a group of smaller EMS's that manage the NE's of the domain. So, the management task is distributed over several entities following some criteria that include functionality, type of errors, or even location. In this setting, the sub-EMS's might need to interact between them and exchange alarm related information. Here, trace analysis could be used to evaluate the behavior of these sub-EMS's. So properties of interest in this context can include the following: how alarms are being managed or in the case of failure -to correlate an alarm correctly, which EMS is not functioning as required.

Note that the successful use of the trace analysis technology in analyzing the behavior of network systems relies on two key issues:

1. Determining the formal model of the interacting NE's. A solution depends on what are the events collected, parameters of the events, relations between events of the same NE and/or different NE (matching communication events) and what is known about the ordering of the events in the trace. Here, we require - from our experience with several application domains- that every event be represented in the collected logfile by a record that shows the following fields:
   - Type of event: Communication (Send, Receive, or Rendezvous), Local
   - Name of the issuing process
   - Local ordinal number
   - Partner for communication events
   - Message / rendezvous parameters
   - Local parameters of the issuing process

2. Scalability of the approach. It is crucial for any approach when applied in industrial settings to be able to handle large amounts of data. In our framework for trace analysis, we have equipped the TRAYSIS tool with filtering techniques that made handling traces of tens of thousands of events much easier. We also focus, in our research, on several techniques to alleviate the problem associated with the size of the monitored traces such as clustering of events (where a set of events sharing certain characteristics could be represented as single event).

4. Example

To illustrate how trace analysis can be used in the context of network management, we consider a simple example of a trace of a hypothetical system of network elements that are monitored by the appropriate means.

In the typical setting, the EMS receives all the monitored information and uses it to build a repository of detailed history of NE specifics (events, states, alarms, etc). Then the operator needs to analyze the stored data to infer the existence of a malfunctioning in the system and perform a root cause analysis. This is fine as long as elements do not interact (each element is affected only by its own behavior), but this is not what networks are meant to be in the first place. In fact, the existence of interactions between elements has a direct effect on identifying the causality relation between events of different elements (sending a data packet could be seen as the cause for the corresponding acknowledgment; a message would not be forwarded if it was not received a priori). Therefore, if an alarm appears on one element, its cause may still be an event of another element that precedes the alarm in the causality precedence relation defined by the interactions between the two elements.

In this example, we use a space time diagram (Figure 2) to show the trace collected by monitoring two elements in a network. The two elements interact by exchanging some messages. This exchange could involve simple send and receive events or, on a more abstract level, a sequence of messages that represent certain functionality such as connection request or file transfer operations. There are three messages in the trace and some local events, which could refer to state updates or indicate alarms raised by the issuing elements. For simplicity, we concentrate on the meaning of the local events rather than the exchanged messages. We consider that each local event contains the value of a variable, which represents the number of alarms raised so far by the issuing element. Thus, the first variable, *n*1, takes the values 3, 4, and finally 5. The second variable, *n*2, takes the values 6, 4, and 2.

Assume now that we have to verify the following: The number of alarms from both NE's is increasing. The EMS manages to correlate the alarms of *NE*2, and the proper measures are taken to reduce them. Does this decrease the number of alarms issued by *NE*1 or not. In other words, do the alarms of *NE*2 constitute a cause of the alarms of *NE*1?

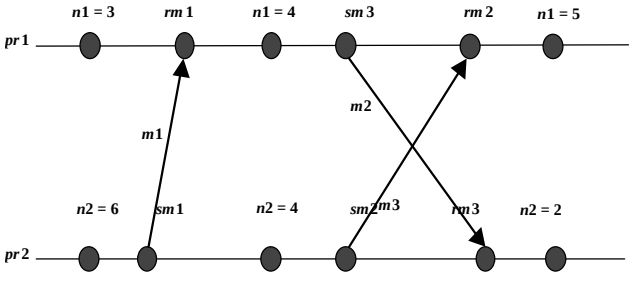

Figure 2: The event sequence of a collected trace, where unconnected bullets correspond to local events, and bullets connected by edges to receive events (*rm*) matching send events (*sm*) for messages *m*1, *m*2, and *m*3.

Assume now that we have to verify the following: The number of alarms from both NE's is increasing. The EMS manages to correlate the alarms of *NE*2, and the proper measures are taken to reduce them. Does this

decrease the number of alarms issued by *NE*1 or not. In other words, do the alarms of *NE*2 constitute a cause of the alarms of *NE*1?

We formulate this requirement as a global property of the system: "Is it possible that the variable *n*2 exceeds the variable *n*1 by 2 or more, i.e., $n2 \geq n1 + 2$, and later *n*1 exceeds *n*2 by 2 or more, i.e., $n1 \geq n2 + 2$?" If this pattern is detected in the trace of the system, it means that the alarms of the two NE's are independent.

One may realize that Figure 2 does not provide an immediate answer to this question. The problem is that events in the system are not totally ordered. Instead, one needs to determine all the possible ordering of events to verify the property. This can be done once each of the communicating processes is modeled, for example, as an automaton. In this case, the behavior of the system of communicating automata contains all the possible event orderings.

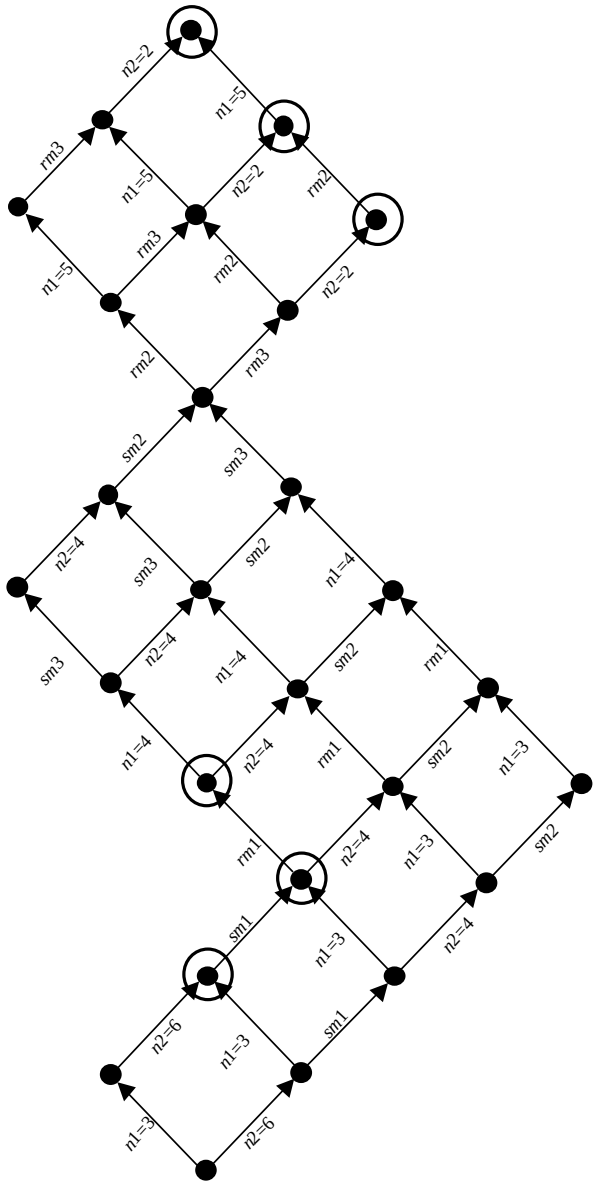

Figure 3: The graph of all the orderings of events of the trace.

In our example, one could define just two linear automata each of which executes a sequence of six events, though the way they communicate must ensure that no execution violates a given trace. We use the TRAYSIS tool to build an SDL model of the two elements and the Property Manager to instantiate an existing pattern of the required property in GOAL. Then, Using OG, the joint behavior of the two automata is depicted in the graph of Figure 3. In this directed graph, it is possible to verify that the property we stated earlier is satisfied in our trace. Actually, the first three encircled nodes (starting from the root) satisfy the first part of the property ($n2 \geq n1 + 2$), while the remaining encircled nodes – the second part ($n1 \geq n2 + 2$). This means that, in term of the behavior of the NE's, the generated alarms are not inter-dependent and need to be handled separately for each element. The model checking of the specified property on the model of the system (by building the synchronous product of the two) confirms this result. The analysis results in OG are shown in Figure 4.

```
Number of states: 44
Number of transitions: 64
Maximum depth reached: 13
Duration : 0 mn 1 s
Number of exceptions: 0
Number of deadlocks: 4
Number of stop conditions: 0
Transitions coverage rate: 100.00
States coverage rate: 100.00
Basic blocks coverage rate: 100.00
Number of errors: 0
Number of success: 3
```

Figure 4: The results of the analysis as shown by ObjectGEODE.

5. Conclusion

In this paper, we proposed an analysis approach for network systems based on formal methods. The trace analysis approach offers a means for the automation of many analysis activities in the network management domain. In addition, such a formal approach based on model checking, guarantees precise analysis results.

This approach relies, primarily, on a tracing and monitoring mechanism that must be integrated with the network system to obtain an execution trace as the starting point of the analysis procedure. Actually, it is quite common that a network system is instrumented, one way or another, to produce that such logfiles. Our implementation considers ObjectGEODE, a model checker by Telelogic, SDL and the GOAL language to model a trace of the system, specify the properties, and perform the verification. We also discussed the main requirements for successful application of the trace analysis approach to network management. We require a clear structure of the events in a trace (providing successful information for modeling and property specification). Also scalability-enhancing techniques are needed to fight the complexity resulting from large traces

being cumulated from hours and days of running of the system. Our current framework ensures a well-defined structure of the trace events, the tools we developed provide filtering techniques that alleviate the problem of lengthy traces in the case of industrial systems. We have applied this approach successfully in many domains, e.g., traffic control systems [15] (safety properties), communication protocol implementations [7] (bandwidth assessment), and wireless mobile networks.

We illustrated, using an example, how this approach can applies to the analysis of network systems, especially in the context of network management, where fault management - in particular, service assurance- is an activity that is error prone and requires a certain level of automation. The main concern for the future is to identify further efficient techniques, in addition to filtering, to enhance the scalability of the approach.

References

[1] Emad Aboelela, Crhistos Douligeris, "Fuzzy Temporal Reasoning Model fro Event Correlation in Network Management", *In Proc. of 24th Confenernce on Local Computer Networks, Massachussetts, USA*, 1999.

[2] B. Algayres, Y. Lejeune, E. Hugonnet, "GOAL: Observing SDL behaviors with GEODE", *in SDL'95 with MSC in CASE (ed. R. Braek, A. Sarma), In Proc. of the 7th SDL Forum*, Oslo, Norway, September, 1995, Elsevier Science Publishers B. V. (North Holland), *pp. 359-372*.

[3] E.M. Clarke, O. Grumberg, and D. Peled. Model Checking. MIT Press, 1999.

[4] M. Dwyer, G. Avrunin, and J. Corbett, "Patterns in Property Specifications for Finite-state Verification", *In Proc. of 21st International Conference on Software Engineering*, May 1999.

[5] Element Management Systems, Online tutorial. http://www.iec.org/online/tutorials/ems/.

[6] R. Groz, "Unrestricted Verification of Protocol Properties on a Simulation Using an Observer Approach", *Protocol Specification, Testing and Verification*, VI, Montréal, Canada, North-Holland, 1986, *pp. 255-266*.

[7] Hallal, H., Boroday, S., Ulrich, A. and Petrenko, A. "An Automata-based Approach to Property Testing in Event Traces" *In Proc. of the IFIP TC6/WG6.1 XV International Conference on Testing of Communicating Systems (TestCom 2003)*. Sophia Antipolis, France, May 26-29, 2003.

[8] Hallal, H., Petrenko, A., Ulrich, A. and Boroday, S. "Using SDL Tools to Test Properties of Distributed Systems". *In Proc. of the Formal Approches to Testing of Software (FATES'01), Workshop of the International Conference on Concurrency Theory (CONCUR'01)*. Aalborg, Denmark, August 21-24, 2001, *pp. 125-140*.

[9] W. Kehl H. Hopfmuller. "Model-based Reasoning for the Management of Telecommunication Networks". *In Proc. of IEEE International Conference on Communications ICC'93*, 1993, *pp. 13-17*.

[10] Meira, D. "A Model For Alarm Correlation in Telecommunications Networks", *PhD Thesis, Federal University of Minas Gerais - UFMG* - Belo Horizonte, Brazil, 1997.

[11] A. Pietschker, A. Ulrich, "A Light-weight Method for Trace Analysis to Support Fault Diagnosis in Concurrent Systems", *In Proc. of the 6th World Multiconference on Systemic, Cypernetics and Informatics (SCI 2002)*; Orlando, FL, USA; July 2002.

[12] Pattern Specification System, online repository: http://www.cis.ksu.edu/santos/spec-patterns.

[13] A. Tanenbaum. "Computer Networks". *Prentice Hall*, 1996.

[14] Telelogic, "ObjectGEODE SDL Simulator Reference Manual", 1999.

[15] A. Ulrich, H. Hallal, A. Petrenko, S. Boroday, "Verifying Trustworthiness Requirements in Distributed Systems with Formal Log-file Analysis". *In Proc. of the thirty-sixth Hawaii International Conference on System Sciences (HICSS-36)*, 2003.

[16] P. Wu, R. Bhatnagar, L. Epshtein, M. Bhandaru, and Z. Shi. Alarm correlation engine (ACE). *In Proc. of IEEE/IFIP Network Operation and Management Symposium*, 1998, *pp. 733--742*.